РОССИЙСКО-АРМЯНСКИЙ УНИВЕРСТИТЕТ

Геворкян Арам Владимирович

РАЗРАБОТКА И РЕАЛИЗАЦИЯ МЕТОДОВ СИНТЕЗА ТРЕХМЕРНЫХ ОБЪЕКТОВ ИЗ ПОСЛЕДОВАТЕЛЬНОСТИ ДВУМЕРНЫХ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ

Диссертация

на соискание учёной степени кандидата технических наук по специальности 05.13.04 – ”Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей”

Научный руководитель: д.ф.м.н. Саруханян А. Г.

Ереван-2019

Содержание

ВВЕДЕНИЕ

Актуальность темы исследования

В настоящее время активно развиваются трехмерные технологии, которые проникли почти во все отрасли нашей жизни. Они используются в киноиндустрии, видео и компьютерных играх, приложениях виртуальной реальности, робототехнике, геоинформационных системах, биометрии, медицине, археологии, военном деле и т.д. Одной из основных задач в данной области является создание трехмерной модели реального физического объекта, которая имеет множество применений и очень актуальна. Данная задача также известна, как задача обратной инженерии.

Создание 3D модели происходит при помощи 3D сканирования. 3D сканер - это устройство, предназначенное для получения трехмерной информации окружающего мира. 3D сканеры разделяются на два типа: контактные и бесконтактные. Контактным 3D сканерам для получения трехмерной информации необходим физический контакт с объектом. Бесконтактные 3D сканеры используют активные или пассивные методы. Активные методы излучают на исследуемый объект направленные волны, а пассивные - анализируют отраженное окружающее излучение. Результатом большинства 3D сканеров является облако точек. Облако точек - это множество точек в некой координатной системе.

Большинство 3D сканеров требуют дорогостоящего оборудования, чувствительны к окружающей среде, освещению и свойствам материалов, из которых сделаны поверхности объектов.

Более доступного решения задачи построения 3D модели объекта можно достичь если использовать стереозрение. Стереозрение является пассивным методом 3D сканирования, который позволяет получить трехмерную информацию при помощи двух камер. Однако для получения 3D модели объекта при помощи стереозрения, необходимо решить задачу совмещения различных ракурсов объекта.

Полученная трехмерная модель имеет множество применений, одним из которых является 3D печать. 3D принтеры активно развиваются начиная с 2000-ых годов и очень актуальны в наше время.

Цель работы

Цель работы разработать метод построения трехмерной модели физического объекта реального мира при помощи двух камер, и на основе разработанного метода реализовать программный комплекс. Данный комплекс должен совмещать методы стереозрения с алгоритмами, работающими с трехмерными данными, решать задачи калибровки камер, стерео сопоставления, построения облака точек, регистрации (совмещения) облаков точек и восстановления поверхности.

Методы исследования

В работе используются методы калибровки камер, ректификации стереоизображений, стерео соответствия (построения карты смещений), обработки и фильтрации облаков точек, регистрации облаков точек и восстановления поверхности из облака точек.

Научная новизна

В работе получены следующие основные результаты:

- Разработан новый метод регистрации облаков точек, который основан на алгоритмах регистрации при помощи FPFH (Fast Point Feature Histograms) дескриптора и ICP (Iterative Closest Point).
- На основе алгоритмов стереозрения и разработанного метода регистрации облаков точек, разработан метод создания 3D модели объекта при помощи двух камер.
- Разработан программный комплекс, который совмещая предложенные методы с известными алгоритмами позволяет получить 3D модель объекта.

Практическая ценность

На основе полученных результатов разработан программный комплекс, который, используя две обычные веб-камеры, позволяет получить трехмерную модель реального физического объекта. Полученная 3D модель может быть использована в 3D графике или быть распечатана при помощи 3D принтера.

Основные положения, выносимые на защиту:

1. Метод регистрации облаков точек, который сначала обрабатывает облака точек (удаляет шумы) и потом для соединения облаков точек использует алгоритм регистрации на основе FPFH дескриптора и алгоритм ICP.
2. Метод получения 3D модели объекта при помощи стереозрения. Данный метод, комбинируя алгоритмы стереозрения с разработанным алгоритмом регистрации облаков точек, получает трехмерную модель объекта.
3. Программный комплекс, который на основе разработанных алгоритмов позволяет получить 3D модель объекта при помощи двух камер.

Внедрение результатов работы

Программный комплекс внедрен и используется компанией "BARVA" для получения 3D модели.

Апробация

Основные результаты и положения диссертационной работы обсуждались и докладывались на семинарах Российско-Армянского Славянского университета и Института проблем информатики и автоматизации НАН РА, представлялись на международной конференции по информационным технологиям и разработки программного обеспечения ITA 2016 в г. Варна, Болгария и на международной конференции «Применение современных научных методов и технологий в области экспертиз» г. Ереван-Цахкадзор, Армения, 2015.

Публикации

Результаты работы отражены в шести публикациях, список которых приведен в конце диссертации.

Структура и объем работы

Диссертация состоит из введения, трех глав, заключения, списка использованной литературы и приложения. Общий объем диссертации составляет 113 страниц.

ГЛАВА 1

МЕТОДЫ ПОЛУЧЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ

1.1 3D СКАНИРОВАНИЕ

3D сканирование - процесс получения трехмерной информации окружающего мира, а устройства, предназначенные для получения трехмерной информации, называются 3D сканерами. 3D сканеры разделяются на два типа: контактные и бесконтактные. Контактным 3D сканерам для получения трехмерной информации необходим физический контакт с объектом. Преимуществами контактных 3D сканеров являются: независимость от уровня освещения, простота использования и высокая точность построенной 3D модели. Преимущество же бесконтактного метода сканирования по отношению к контактному состоит в том, что не требуется непосредственного соприкосновения с объектом, что дает возможность сканирования труднодоступных или крупномасштабных объектов, например, памятников архитектуры. Бесконтактные 3D сканеры, в свою очередь, делятся на две группы: активные и пассивные.

Активные сканеры излучают на исследуемый объект направленные волны, чаще всего лазерный луч или структурированный свет, после чего обнаруживают и анализируют их отражение. К активным методам сканирования относятся: лазерные сканеры, времяпролетные (time-of-flight) камеры. Стандартные подходы используют лазерные луч или структурированный свет, новейшие методы чаще используют Лидар (Lidar) [1] технологию. Активные методы сканирования дают точный результат, но они также требуют дорогостоящего оборудования и чувствительны к свойствам материалов, из которых сделаны поверхности объектов. Также на них влияет активное освещение, что делает их неприемлемыми в неконтролируемых или многолюдных средах.

Пассивные методы сканирования на объект ничего не излучают и анализируют отраженное окружающее излучение, чаще всего видимый свет. К пассивным методам

относятся: фотометрический метод, структура из тени (structure-from-shading), структура из текстуры (structure-from-texture), структура из движения (structure-from-motion), стереозрение (stereo vision). Фотометрический метод, использует набор изображений, полученных при освещении объекта различными источниками света. Структура из тени использует тени на изображениях, полученные с различных ракурсов и при различных условиях освещения, которые содержат информацию о форме объекта. Структура из движения (Structure From Motion) используется для нахождения трехмерной структуры объекта из анализа локального движения частей сцены с течением времени. Стереозрение - это метод получения трехмерной информации объекта, при помощи двух или более видео камер. Более подробно стереозрение рассмотрим в следующих разделах.

В работе [2] приведен обзор различных 3D сканеров и техник 3D сканирования, а также их сильные и слабые стороны.

Существуют также подходы, которые совмещают активные и пассивные методы, например, используя камеры и структурированный свет или лазерный луч. В работе [3] описано построение 3D сканера на основе двух камер и структурированного света, а в работе [4] на основе стереокамеры и лазерного луча.

Результатом 3D сканирования обычно являются множество отдельных точек неизвестной поверхности (облако точек), что является неприемлемым для множества приложений, поэтому к множеству точек, применяется один из алгоритмов восстановления поверхности (surface reconstruction). Более подробно алгоритмы восстановления поверхности рассмотрены в разделе 1.10.

3D модель может быть использована в различных приложениях компьютерной графики, видеоиграх, кино, а также для 3D печати (более подробно о 3D печати в разделах 1.10, 3.11).

1.2 СТЕРЕОЗРЕНИЕ

Стереозрение - одно из основных направлений компьютерного зрения, используется при составлении трехмерной модели окружающего мира и позволяет получить информацию о глубине изображения, формах, размерах и расстояниях до объектов, при помощи двух или более изображений, отснятых с разных ракурсов. Основной задачей стереозрения является построение карты глубины (depth map). Карта глубины из себя представляет информацию, где каждой точке на изображении соответствует ее расстояние до камеры (глубина). Карта глубины может быть преобразована в другие форматы, хранящие трехмерные данные, например, облако точек. Более подробно облако точек описано в разделе 1.9.

Традиционное стереозрение использует две камеры для получения карты глубины, что похоже на эффект бинокулярного зрения. Бинокулярного зрение - это человеческая способность видеть предмет объемно, при помощи левого и правого глаза. Пара изображений, отснятых левой и правой камерой, называется стереопарой или стереоизображением. Один и тот же предмет на левом и правом изображении имеет различное угловое смещение, этот эффект называется *параллаксом*. Основная идея заключается в нахождении соответствующих точек между левым и правым изображениями, вычислении разницы их координат (построение карты смещений - disparity map) и на основе этой разницы построении карты глубины. Таким образом, построение карты глубины состоит из следующих этапов:

- *калибровка* - нахождение параметров камер и удаление искажений;
- *ректификация* - выравнивание изображений к общей плоскости;
- *стерео-соответствие* - нахождение соответствующих точек и построение карты смещений;
- *триангуляция* - построение карты глубины по карте смещений.

Каждый из этих этапов подробно рассмотрен в следующих разделах.

В работе [5] приведены современные системы стереозрения. Существуют также методы стереозрения, которые позволяют получать трехмерную информацию при помощи одной камеры [6], а также трех или более камер (см. [7], глава 12.3). При использовании одной камеры, сцена фотографируется камерой с двух разных положений. Зная, как они геометрически связаны между собой, можно применить тот же алгоритм, что и при использовании двух камер. В случае использования трех камер, изображение с третьей камеры обычно используется для проверки соответствий между изображениями, полученными с помощью первой и второй камеры.

1.3 ПОЛУЧЕНИЕ СТЕРЕОПАРЫ

Для получения трехмерной информации стереозрение использует стереопару изображений. Существуют различные способы получения стереопары. В 1950-х появилась идея размещать два объектива на одной камере, в те года была популярна камера Stereo Realist, снимавшая стереоизображения на 35-и миллиметровую плёнку [8]. В настоящее время существуют цифровые стерео-видеокамеры и фотоаппараты, работающие по аналогичной концепции: на корпусе размещаются два объектива на расстоянии 64мм друг от друга, соответствующий среднему межглазному расстоянию у человека. К самым популярным стереокамерам относятся: Bumblebee XB3 и Bumblebee 2 [9], Kinect 3D [10], Surveyor Stereo Vision System (“SVS”) [11] - система стерео-зрения с открытым доступом, разработанная специально для обучающих и научных целей, MEGA-DCS [12], PCI nDepth vision system [13] и Minoru 3D Webcam [14] - дешевая стереокамера в основном для студенческих проектов. Также существуют смартфоны и планшеты, которые имеют две встроенные камеры, что дает возможность использовать их в качестве стереокамеры. Например, Iphone 7 Plus имеет встроенные две основные камеры: первая с широкоугольным объективом 56мм и диафрагмой (aperture) *f/1.8*, а вторая с телескопическим объективом 56мм и диафрагмой *f/2.8*. В работе [15] описан метод получения карты глубины для планшета NVIDIA Tegra 3.

Альтернативой вышеуказанным специализированным устройствам, которые в основном дорогостоящие, являются программно-аппаратные системы, решающие задачи калибровки и синхронизации для произвольных наборов камер. Для облегчения задачи калибровки и синхронизации, лучше всего использовать две одинаковые веб-камеры, закрепленные на фиксированном расстоянии друг от друга.

Существуют также интернет ресурсы с отснятым множеством стереоизображений и некоторыми параметрами камер [16-20]. Сайт института Middlebury [21] специально создан для тестирования и сравнения алгоритмов стерео сопоставления, где находятся множество баз данных со стереопарами и необходимыми параметрами двух камер.

На Рисунке 1.1 приведен пример известной стереопары Tsukuba [22].

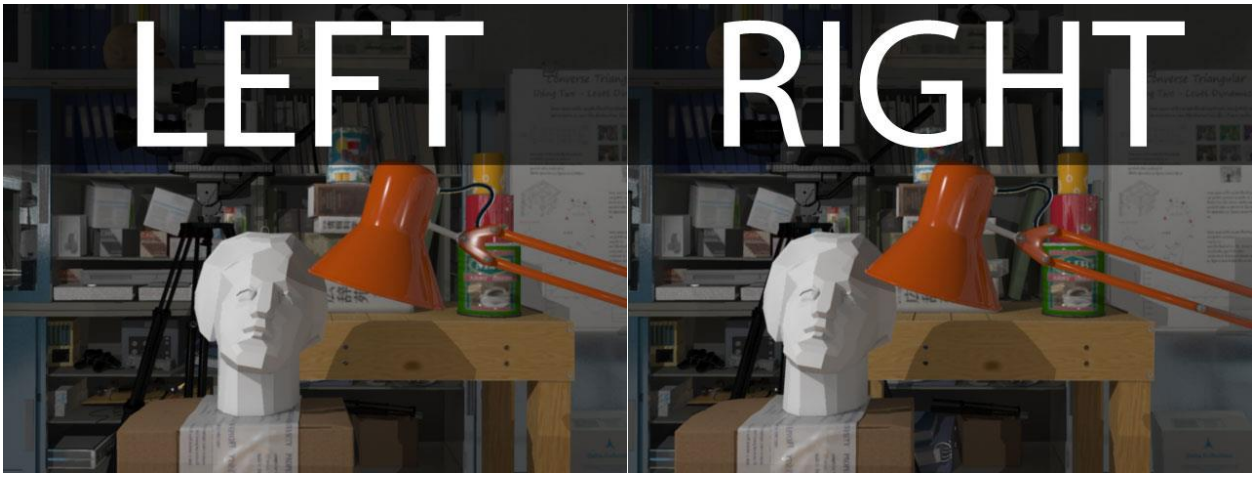


Рисунок 1.1: Стереопара Tsukuba.

1.4 КАЛИБРОВКА КАМЕРЫ

Калибровка камеры [23] - одна из основных и важнейших задач стереозрения, которая необходима для ректификации, вычисления карты глубины и удаления искажений (дисторсий) на фотографиях и видео. Калибровка камеры - это процесс вычисления внешних и внутренних параметров камеры по фотографиям или по видео, сделанных ею. Внутренние параметры содержат такие данные, как фокусное расстояние, размер пикселя, координаты принципиальной точки и коэффициент дисторсии. Внешние параметры показывают позицию и ориентацию камеры в пространстве. Стерео калибровка камер - это вычисление внутренних параметров обеих камер, а также геометрического расположения камер по отношению друг к другу.

Калибровка камеры также обычно решает проблему искажений. Так как оптика не идеальна, то на изображениях, полученных с помощью камер, присутствуют искажения (дисторсии, distortion). Искажения в основном можно разделить на два типа: радиальные и тангенциальные. Радиальные искажения вызваны неидеальностью параболической формы линзы, а тангенциальные - погрешностями в установке линзы параллельно плоскости изображения. На рисунке 1.2 показан пример искажений: a) - показывает, как влияют искажения на проектирование точки, b) - показывает эффект положительных и отрицательных радиальных искажений, c) - показывает тангенциальные искажения (прямые линии - когда нет дисторсий, пунктирные линии - это когда есть тангенциальные дисторсии).

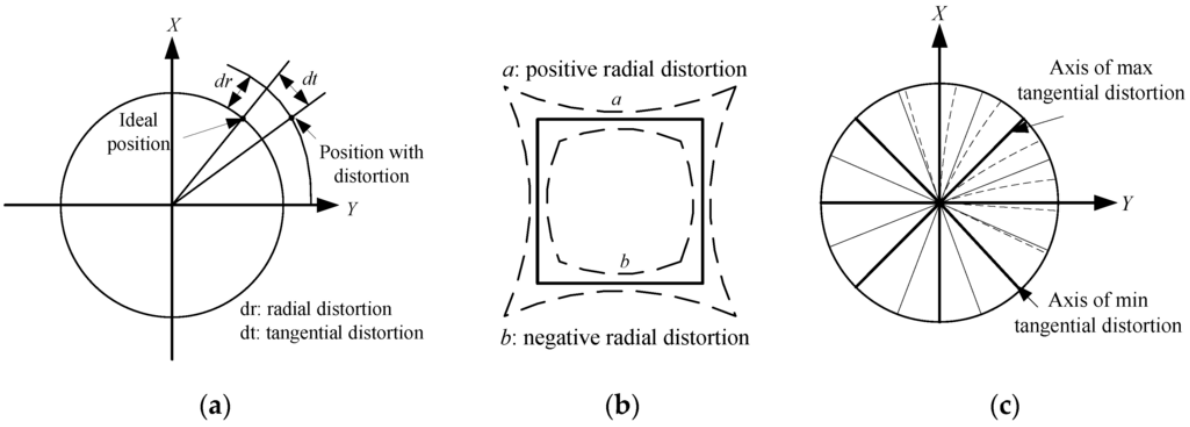


Рисунок 1.2: Пример искажений [24].

Для устранения искажений применяется следующее преобразование координат пикселей [24]:

$$U_{cor} = u(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 uv + p_2(r^2 + 2u^2),$$

$$V_{cor} = v(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_2 uv + p_1(r^2 + 2v^2),$$

где (u, v) - первоначальные координаты пикселя, (u_{cor}, v_{cor}) - координаты пикселя после устранения искажений, k_1, k_2, k_3 - коэффициенты радиальных искажений, p_1, p_2 - коэффициенты тангенциальных искажений, $r^2 = u^2 + v^2$.

Для понимания работы камеры и описания ее параметров рассмотрим модель камеры-обскуры.

Камера-обскуры (pinhole camera) - это простейшая модель камеры, которая используется для описания принципа работы современных фотоаппаратов и камер. Камера-обскуры [25] определяется *центром камеры*, *принципиальной осью* - лучом, начинающимся в центре камеры и направленным по направлению камеры, *плоскостью изображения* - плоскостью на которую выполняется проецирование точек, и *системой координат* на этой плоскости. На рисунке 1.3 показана модель камеры-обскуры.

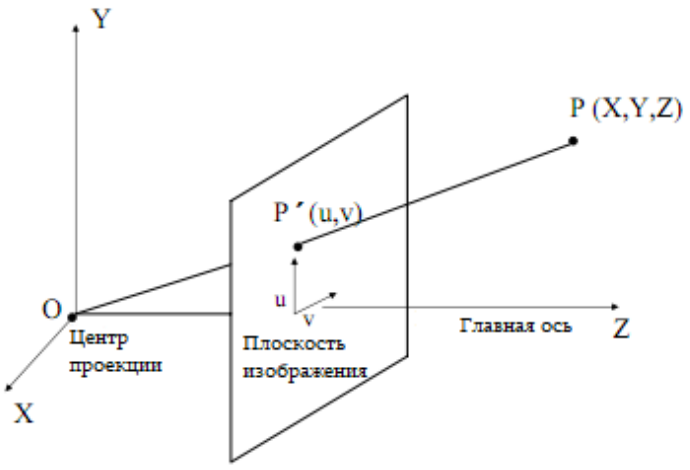


Рисунок 1.3: Модель камеры-обскуры [25].

На рисунке 1.3 $O(O_x,O_y)$ - это центр камеры (проекции); f - фокусное расстояние (расстояние от плоскости изображения до центра камеры).

Главная ось - это линия перпендикулярная плоскости изображения и проходящая через точку O. Пересечение главной оси с плоскостью изображения, называется *принципиальной точкой.*

Точке *реального мира* $P(X,Y,Z)$ соответствует точка на *плоскости изображения* $P'(U,V)$, при помощи следующего проективного преобразования:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = C \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

В простейшем случае, когда центр камеры находится в точке $(0,0,0)$, а ось Z перпендикулярна плоскости изображений преобразование имеет следующий вид:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K(R \mid T) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \text{ где } K = \begin{pmatrix} A_x & s & x_0 \\ 0 & A_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}, R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, T = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

A_x и A_y - фокусные расстояния по направлению x и y; (x_0, y_0) – координаты принципиальной точки; z - параметр сжатия; K - матрица внутренних параметров; R - матрица вращения; T - вектор трансляции. R и T - внешние параметры камеры.

Если известны матрицы внутренних и внешних параметров камеры, то задача калибровки решена.

Существуют различные методы калибровки, к которым относятся:

- Алгоритм Роджер Цайя (Roger Y. Tsai) [26], который состоит из двух этапов: на первом этапе определяются внешние параметры камеры, а на втором - внутренние параметры и дисторсии (искажения). Данный метод основан на

сопоставлении координат точек на калибровочном объекте и на плоскости изображения. В качестве калибровочного объекта используется объект, состоящий из двух или трех ортогональных плоскостей. Иногда также используется движущаяся плоскость с известным перемещением. Трудности в реализации алгоритма связаны с построением калибровочного объекта. Алгоритм Tsai учитывает только радиальные искажения.

- Алгоритм Чжэн Жанга (Zhengyou Zhang) - "Новая гибкая технология калибровки камеры" [27], который основан на использовании плоского калибровочного объекта в виде шахматной доски. Для работы алгоритма используется несколько (в основном 15) снимков калибровочного объекта, отснятого с разных ракурсов и положений. Данный метод прост для реализации и является наиболее часто используемым.
- Автокалибровка - получение параметров камеры непосредственно по изображениям, при этом не требуется наличия специальных калибровочных объектов. Автокалибровка является более трудоемкой задачей и обычно дает менее точные результаты. Всегда, когда возможно, предпочтительно использовать калибровку камеры при помощи калибровочного объекта. Более подробное описание алгоритма автокалибровки приведено в книге [28] Глава 19.

Существуют программные инструменты, которые решают задачу калибровки. Например, в [29] описан инструмент на основе Mathlab, который калибрует камеры по заданным изображениям шахматной доски с разных ракурсов.

1.5 РЕКТИФИКАЦИЯ

Если две камеры параллельны друг другу, то их изображения имеют одинаковую плоскость изображений и тогда соответствующие точки левых и правых изображений находятся на одной строке. Тем самым для поиска соответствующей точки достаточно рассматривать не всё изображение, а только соответствующую строку. Однако в реальных условиях установить две камеры идеально параллельно друг другу

невозможно, поэтому необходима ректификация. Ректификация [30] (rectification, корректирование) - это процесс проектирования изображений стереопары на плоскость, параллельную линии, которая соединяет центры объективов двух камер (см. Рис. 1.4).

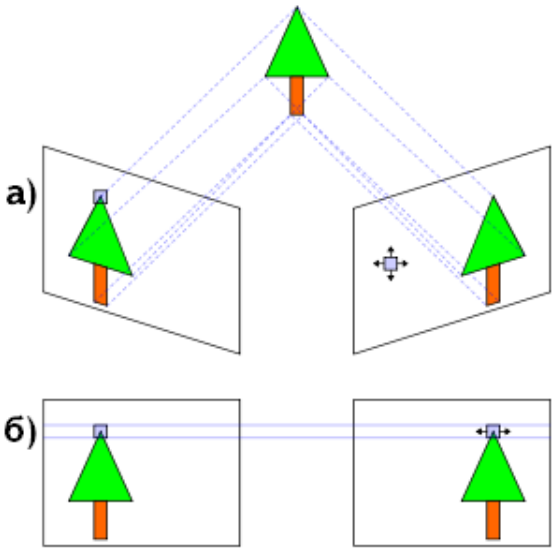


Рисунок 1.4: Плоскости изображений камер до (а) и после (б) ректификации [30].

Для ректификации изображений, а также поиска соответствующих точек применяется эпиполярная геометрия. Эпиполярная геометрия - это проективная геометрия, которая показывает связь между двумя плоскостями изображений.

Рассмотрим ниже основные понятия эпиполярной геометрии.

Эпиполюсы - это точки пересечения базовой линии (линия, которая соединяет центры камер) с плоскостями изображений.

Эпиполярная плоскость - это плоскость, которая формируется при помощи точки трехмерного пространства и двух центров проецирования (точки, в которых расположены камеры).

Эпиполярная линия - пересечение эпиполярной плоскости с плоскостями изображений.

Вышеуказанные определения показаны на рисунке 1.5.

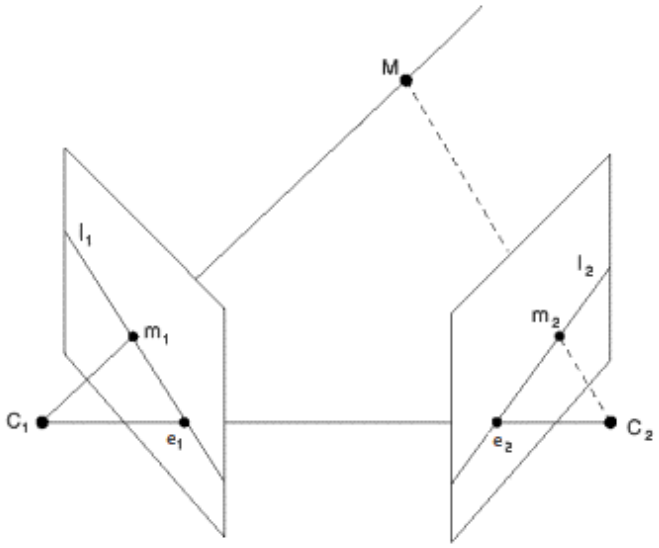


Рисунок 1.5: Эпиполярная геометрия: C_1, C_2 - центры камер, C_1C_2 - базовая линия, M - точка трехмерного пространства, m_1, m_2 - отображения точки в левой и правой плоскостях изображений, e_1, e_2 - эпиполюсы, l_1, l_2 - эпиполярные линии, C_1C_2M - эпиполярная плоскость.

Эпиполярное ограничение: Для точки левого изображения соответствующую ей точку необходимо искать на соответствующей эпиполярной линии правой плоскости изображения, и наоборот. Таким образом, для точки m_1 соответствующую ей точку m_2 нужно искать на эпиполярной линии l_2 .

Алгебраически это ограничение записывается следующим образом:

Если m_1 и m_2 соответствующие точки, то существует такая матрица F размера 3 на 3 и рангом равным 2, что:

$$m_2^T F m_1 = 0,$$

где преобразование Fm_1 ставит в соответствие точке из левого изображения соответствующую эпиполярную линию из правого. Матрица F - называется фундаментальной матрицей. Если известно, как минимум 8 соответствующих точек

между плоскостями изображений, то решая множество линейных уравнений можно вычислить фундаментальную матрицу. Более подробно как вычислять фундаментальную матрицу и о ректификации описано в работе [31].

Таким образом, после ректификации стереопары изображений, соответствующие эпиполярные линии параллельны и соответствующие точки лежат на одной и той же строке в обоих изображениях.

1.6 СТЕРЕО СООТВЕТСТВИЕ

Задача стерео соответствия (или стерео сопоставления) является одной из ключевых задач стерео зрения. Суть задачи стерео соответствия заключается в поиске соответствующих точек левого и правого изображений. После ректификации задача поиска соответствующих точек упрощается. Пусть (x, y) координаты точки на левом изображении, а (x', y') на правом, то после ректификации $y = y'$, т. е. соответствующую пару необходимо искать в той же строке изображения. Величина $x - x'$ называется смещением (диспаритет, disparity). Цель алгоритмов стерео соответствия заключается в построении карты смещений. На рисунке 1.6 показаны соответствующие точки двух изображений, а на рисунке 1.7 - построенная по ним карта смещений.

Рисунок 1.6: Соответствующие точки на левом и правом изображениях [21].

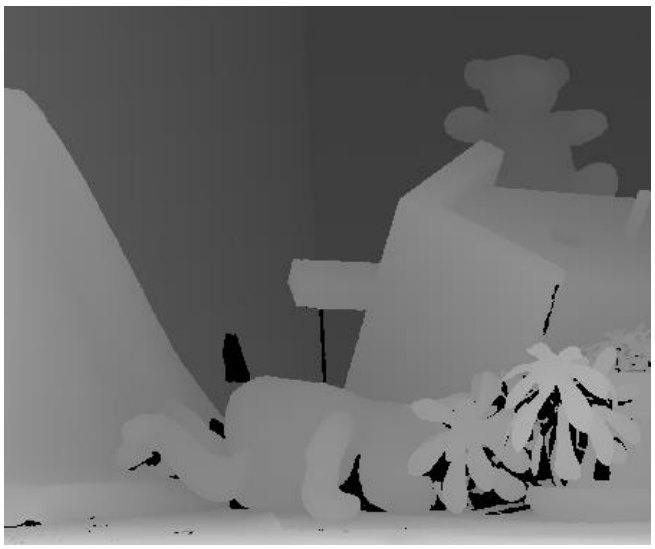


Рисунок 1.7: Карта смещений [21].

Кроме ректификации, большинство алгоритмов стерео сопоставления подразумевают некоторые исходные предположения:

- Ламбертовость поверхностей - независимость освещения поверхности от угла зрения. В основном, алгоритмы стерео соответствия не учитывают особенности распространения света.
- Поверхности в реальном мире кусочно-гладкие. Если соседние пиксели имеют различные смещения или интенсивности, то алгоритмы стерео соответствия обычно налагают штраф.
- Многие алгоритмы предполагают, что на границах объектов интенсивность пикселей меняется достаточно резко.

Задача стерео соответствия может осложнятся некоторыми следующими факторами:

- Из-за несовершенности сенсоров, а также из-за плохого освещения на фотографиях может присутствовать шум.
- Присутствие однотонных областей на изображениях, что осложняет задачу поиска соответствующего пикселя.
- Некоторые области или пиксели могут присутствовать только на одном из изображений.

Существует множество алгоритмов стерео сопоставления. Детальное описание алгоритмов можно найти в работе [32]. В работе [33] приведен сравнительный анализ современных алгоритмов стерео соответствия. В основном алгоритмы стерео соответствия можно разделить на две группы: локальные и глобальные.

Локальные алгоритмы при вычислении смещения для каждой точки, используют информацию только из окрестности этой точки (квадратная или прямоугольная область вокруг данной точки).

Глобальные алгоритмы, в отличии от локальных, не вычисляют смещение для каждого пикселя отдельно, а целью глобальных методов является поиск наилучшей карты смещений для всего изображения. Обычно глобальные алгоритмы стремятся минимизировать функцию энергии:

$$E(f) = E_{data}(f) + E_{smooth}(f),$$

где $E_{data}(f)$ показывает насколько хорошо данная конфигурация согласуется с парой входных изображений, а $E_{smooth}(f)$ показывает штраф за нарушение непрерывности смещений.

Задача минимизации функции энергии может быть решена при помощи динамического программирования [34].

Локальные алгоритмы более эффективны по сравнению с глобальными, однако качество у глобальных алгоритмов лучше.

В последнее время приобретают популярность методы, пытающиеся совместить сильные стороны локальных и глобальных алгоритмов. Решение в каждом пикселе принимается независимо, однако влияние на него оказывают либо все остальные пиксели изображения, либо часть пикселей, не ограниченных локальной окрестностью данной точки. Одним из таких алгоритмов является алгоритм полу-глобального сопоставления (Semi-Global Matching), который рассмотрен в следующем параграфе.

1.7 ОПИСАНИЕ АЛГОРИТМА SEMI-GLOBAL MATCHING (SGM)

Алгоритм полуглобального сопоставления (Semi-Global Matching - SGM) [35] является одним из наиболее оптимальных алгоритмов стерео сопоставления, так как он в себе сочетает качество глобальных подходов и скорость локальных. Алгоритм был предложен в 2005 году Heiko Hirschmuller-ом.

Алгоритм состоит из следующих шагов:

1) Вычисление попиксельной стоимости.
2) Суммирование попиксельных стоимостей.
3) Вычисление карты смещений.

Рассмотрим данные шаги более подробно.

1) ВЫЧИСЛЕНИЕ ПОПИКСЕЛЬНОЙ СТОИМОСТИ

Пусть мы хотим вычислить попиксельную стоимость C для пикселя $p=(p_x, p_y)$ и смещения d между левым и правым изображениями (I_l и I_r). Существуют различные способы вычисления попиксельной стоимости. Рассмотрим некоторые из них.

- Метод абсолютной разницы:

$$C(p,d)=\left|I_l(p_x,p_y)-I_r(p_x-d,p_y)\right|,$$

- Вычисление попиксельной стоимости на основе взаимной информации:

Обозначим в данном случае попиксельную стоимость через $C_{MI}(p,d)$ и для начало рассмотрим функцию энергии, которая вычисляется по формуле:

$$E(f)=E_{data}(f)+E_{smooth}(f)$$

$E_{data}(f)$ - штраф за несогласование с исходными данными, $E_{smooth}(f)$ - штраф для конфигураций, нарушающих ограничение непрерывности глубины.

В общем случае:

$$E_{data}(f) = \sum_{p} C(p, f(p)),$$

где f – карта смещений.

В данном случае: $E_{data}(f) = MI(I_l, I_r, f)$,

где MI - взаимная информация между левым и правым изображениями при данной карте смещений f. Взаимная информация зависит от энтропии левого и правого изображения и совместной энтропии двух случайных величин. В данном случае, случайные величины - это пиксели изображений.

Взаимная информация имеет вид: $MI(I_l, I_r, f) = H_l + H_r - H_{lr}$,

где H_l - энтропия левого изображения, H_r - энтропия правого изображения, H_{lr} - совместная энтропия, которые определяются по формулам:

$$H_l = -\int_0^1 P_l(i)\log(P_l(i))di\ ,\ H_r = -\int_0^1 P_r(i)\log(P_r(i))di\ ,$$

$$H_{lr} = -\int_0^1\int_0^1 P_{lr}(i_1, i_2)\log(P_{lr}(i_1, i_2))di_1 di_2$$

Воспользовавшись методом Парзена [36] и взяв в качестве ядра Гауссово распределение, получим:

$$H_l = \sum_p h_l(I_l(p)),\quad H_r = \sum_p h_r(I_r(p)),\quad H_{lr} = \sum_p h_{lr}(I_l(p), I_r(p)),$$

где

$$h_l(i) = -\frac{1}{N}(\log(P_l(i) \otimes g(i) \otimes g(i))),$$

$$h_r(i) = -\frac{1}{N}(\log(P_r(i) \otimes g(i) \otimes g(i))),$$

$$h_{lr}(i,k) = -\frac{1}{N}(\log(P_{lr}(i,k) \otimes g(i,k) \otimes g(i,k))),$$

$$P_{lr}(i,k) = \frac{1}{N}\sum_{p} T\left[(i,k) = (I_l(p), I_r(p - f(p)))\right],$$

$$P_l(i) = \sum_k P_{lr}(i,k),\ P_r(i) = \sum_i P_{lr}(i,k),$$

где f - карта смещений, $T[x]$ - принимает значение 0 если выражение x ложно и 1 в противном случае.

Таким образом, попиксельная стоимость для пикселя p и смещения d равна:

$$C_{MI}(p,d) = h_{lr}(I_l(p), I_r(p-d)) - h_l(I_l(p)) - h_r(I_r(p-d))$$

Так как для вычисления попиксельной стоимости необходимо иметь карту смещений, то на первом шаге выбирается случайная карта смещений (например, заполненная нулями). Далее, после вычисления стоимости, для каждого пикселя находится то смещение, при котором достигается наименьшая попиксельная стоимость, записывается значение этого смещения в карту смещений и заново вычисляется попиксельная стоимость. На практике, трех итераций алгоритма вполне достаточно для получения хорошей карты смещений.

Таким образом, *алгоритм вычисления попиксельной стоимости на основе взаимной информации* состоит из следующих шагов:

1. Берется в качестве первого приближения карты смещений нулевая матрица.
2. Вычисляется совместное распределение по формуле:

$$P_{lr}(i,k) = \frac{1}{N}\sum_{p} T\left[(i,k) = (I_l(p), I_r(p - f(p)))\right]$$

3. Вычисляется распределение для левого и правого изображения, используя совместное распределение:

$$P_l(i) = \sum_k P_{lr}(i,k),\ P_r(i) = \sum_i P_{lr}(i,k)$$

4. Далее вычисляется:

$$h_l(i) = -\frac{1}{N}(\log(P_l(i) \otimes g(i) \otimes g(i))),$$

$$h_r(i) = -\frac{1}{N}(\log(P_r(i) \otimes g(i) \otimes g(i))),$$

$$h_{lr}(i,k) = -\frac{1}{N}(\log(P_{lr}(i,k) \otimes g(i,k) \otimes g(i,k)))$$

5. Вычисляется попиксельная стоимость по формуле:

$$C_{MI}(p,d) = h_{lr}(I_l(p), I_r(p-d)) - h_l(I_l(p)) - h_r(I_r(p-d))$$

6. Если достигнута нужная карта смещений (например, 3-ая итерация), то алгоритм завершается. В противном случае для каждого пикселя находится, то смещение на котором достигается наименьшая попиксельная стоимость и соответствующее значение записывается в текущую карту смещений и алгоритм продолжается с шага 2.

- Вычисление попиксельной стоимости методом Birchfield-Tomasi [37]:

Обозначим через I_r^- - линейно интерполированную интенсивность между y_i и пикселем слева, аналогично I_r^+ - между y_i и пикселем справа, которые определяются по формуле:

$$I_r^- = \frac{1}{2}(I_r(y_i) + I_r(y_i - 1)),\ I_r^+ = \frac{1}{2}(I_r(y_i) + I_r(y_i + 1))$$

Обозначим:

$$I_{\min} = \min(I_r^-, I_r^+, I_r(y_i)),$$

$$I_{\max} = \max(I_r^-, I_r^+, I_r(y_i)),$$

$$f(x_i, y_i, I_l, I_r) = \max\{0, I_i(x_i) - I_{\max}, I_{\min} - I_l(x_i)\}$$

Тогда попиксельная стоимость определяется формулой:

$$C_{BT}(p,d) = f(p, p-d, I_l, I_r).$$

2) СУММИРОВАНИЕ ПОПИКСЕЛЬНЫХ СТОИМОСТЕЙ

После того как вычислены попиксельные стоимости, необходимо их суммировать таким образом, чтобы минимизировать функцию энергии, которая определяется формулой: $E(f) = E_{data}(f) + E_{smooth}(f)$. Классическим оптимизационным методом ее решения является динамическое программирование, но в данном случае суммирование происходит только по одному направлению, поэтому связи между пикселями в разных рядах практически не учитываются. В работе [35] предлагается суммировать по 16 различным направлениям, однако для оптимизации алгоритма можно суммировать по 8 направлениям, качество результирующей карты смещений изменяется не сильно, но скорость увеличивается в 2 раза.

Сумма стоимостей по направлению *r* для пикселя *p* и смещения *d* определяется формулой:

$$L_r(p,d) = C(p,d) + \min(L_r(p-r,d), L_r(p-r,d-1) + P_1, L_r(p-r,d+1) + P_1, \min_i(L_r(p-r,i)) + P_2)$$

Где $C(p,d)$ - попиксельная стоимость, P_1 - штраф налагаемый в том случае, если у соседних пикселей смещение отличается на 1, P_2 - штраф, если у соседних пикселей смещение отличается больше чем на 1. Суммирование происходит рекурсивно.

Значения L_r постоянно растут, что может привести к большим значениям. Поэтому следует модифицировать данное выражение:

$$L_r(p,d) = C(p,d) + \min(L_r(p-r,d), L_r(p-r,d-1) + P_1, L_r(p-r,d+1) + P_1, \min_i(L_r(p-r,i)) + P_2) - \min_k(L_r(p-r,k))$$

Вычитаемое слагаемое постоянно для всех смещений пикселя *p*, таким образом минимум не изменился, зато значение L_r теперь не будет превышать $C_{\max} + P_2$.

После вычисления суммарных стоимостей для каждого из направлений, общая сумма всех стоимостей для каждого пикселя будет равна:

$$S(p,d) = \sum_r L_r(p,d)$$

3) ВЫЧИСЛЕНИЕ КАРТЫ СМЕЩЕНИЙ

После суммирования попиксельных стоимостей карта смещений может быть вычислена следующим образом. Для каждого пикселя p левого изображения, соответствующее ему смещение определяется как:

$$f(p) = \min_d S(p,d)$$

1.8 ВЫЧИСЛЕНИЕ КАРТЫ ГЛУБИНЫ ПРИ ПОМОЩИ ТРИАНГУЛЯЦИИ

Если известны параметры калибровки камер и карта смещений, то при помощи триангуляции можно вычислить карту глубины [38,39]. Допустим имеем две камеры с параллельными оптическими осями - этого всегда возможно достичь при помощи ректификации. Пусть базовая линия перпендикулярна к оптическим осям камер и параллельна к оси X, как показано на рисунке 1.8.

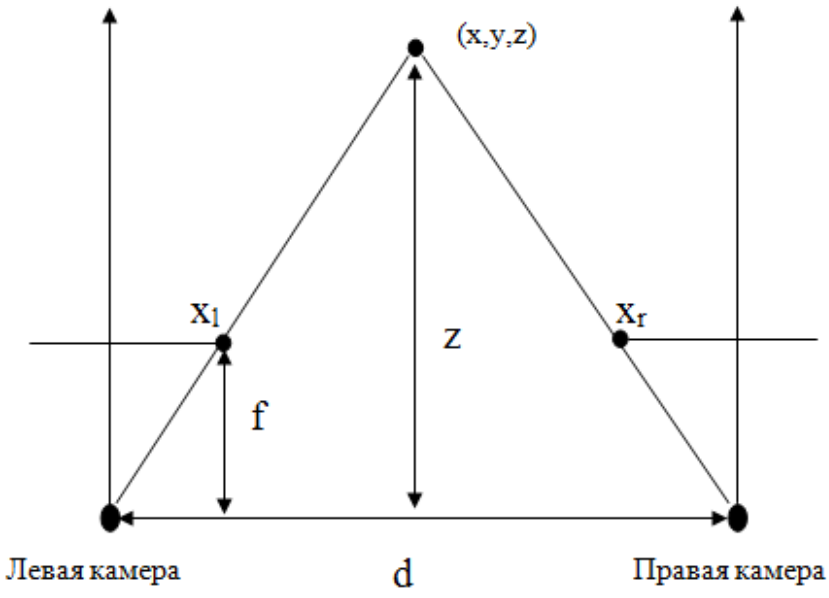


Рисунок 1.8: Вычисление карты глубины

Расстояние между камерами равно d , а фокусное расстояние для обеих камер - f .

Пусть точка в реальном мире имеет трехмерные координаты (x, y, z), где z есть глубина, которую нам надо вычислить. Пусть (x_l, y_l) и (x_r, y_r) координаты этой точки на левой и правой плоскости изображений соответствующих камер.

Тогда из равенства подобных треугольников следует:

$$\frac{z}{d} = \frac{f}{x_l - x_r} \Rightarrow z = \frac{df}{x_l - x_r}$$

где $x_l - x_r$ есть смещение. Из равенства видно, что глубина обратно пропорциональна смещению. Поэтому глубина вычисляется с лучшей точностью для объектов расположенных ближе к камере.

Значения координат x и y вычисляются по формулам:

$$x = \frac{d(x_l + x_r)}{2(x_l - x_r)}, \qquad y = \frac{d(y_l + y_r)}{2(x_l - x_r)}$$

Один из способов представления карты глубины - это *облако точек*, которое описано в следующем разделе.

1.9 ОПИСАНИЕ ОБЛАКА ТОЧЕК

Облако точек (point cloud) - это структура данных, которая содержит многомерную информацию или другими словами множество точек в некой системе координат. Облако точек в основном используется для описания внешней поверхности объекта. В трехмерной системе координат X, Y и Z положение точек облака определяется заданием координат x, y и z. Облако точек также может содержать другую дополнительную информацию о точках: цвет и направление нормали. Например, цветовая информация содержится в виде RGB значения, тогда точка вместе с ее трехмерными координатами и цветом, будет представлена в виде шестикомпонентного вектора $<x,y,z,r,g,b>$.

Облако точек используются для построения 3D моделей, а также в множестве различных приложений, связанных с компьютерной графикой и анимацией. Оно имеет приложения в робототехнике для навигации и восприятия роботов, стереозрении и современных системах содействия водителю (ADAS). Облако точек также используются в промышленной метрологии и инспекции, а также в медицине. В географической информационной системе, облака точек являются одним из источников для создания цифровых моделей рельефа местности. Облака точек используются также для того, чтобы генерировать 3D модели городской среды.

Облако точек может быть получено сразу, как непосредственный результат 3D сканера, например, камеры Microsoft Kinect, а также может быть преобразовано из других форматов специальным программным обеспечением. Так же существуют интернет ресурсы, которые содержат облака точек. [40] - сайт Стенфордского университета, где содержатся примеры облаков точек, которые используются во многих статьях и для различных экспериментов.

Наиболее используемые форматы файлов, которые применяются для хранения облаков точек:

- VTK (visualization toolkit files) - используются для хранения различных данных 3D графики, в том числе, облаков точек.
- CSV (comma separated values) - самый простой формат файла для хранения облаков точек, где значения записываются отделенные запятой. Дата (информация) внутри файла структурирована в виде таблицы, где столбцы отделяются запятой, а строки tab-ом или точкой запятой.
- LAS (laser file format) - формат файла, который обычно содержит облака точек, полученные при помощи лазерного 3D сканера.
- PLY (polygon file format) - формат файла был разработан Стэндфордской графической лабораторией, для хранения данных трехмерной графики. PLY файлы содержат заголовочную часть, которая описывает элементы и свойства используемые в файле. Остальная часть содержит данные, которые могут хранится, как в виде простого текста (ASCII), так и в бинарном виде.
- PCD (point cloud library file format) - формат файла, разработанный библиотекой PCL специально для хранения облаков точек. Содержит заголовочную часть, которая описывает облако точек и данные, которые могут быть записаны в бинарном виде или в виде ASCII.

На практике часто бывает нужно соединить несколько облаков точек в одно общее облако точек. Это могут быть облака точек различных пересекающихся сцен или облака точек различных ракурсов объекта. Эти облака точек нужно привести в одну общую координатную систему. Данный процесс называется 3D регистрацией и более подробно будет рассмотрен в Главе 2.

В большинстве 3D приложений облако точек непосредственно не используется, а преобразуется в полигон или модели треугольников или CAD (computer-aided design) модель. Данный процесс называется 3D реконструкцией или восстановлением поверхности.

1.10 ВОССТАНОВЛЕНИЕ ПОВЕРХНОСТИ ИЗ ОБЛАКА ТОЧЕК ДЛЯ 3D ПЕЧАТИ

Большинство 3D сканеров, получают трехмерную информацию в виде облака точек, но этого бывает недостаточно для множества приложений (например, для 3D печати), часто бывает необходимо построить первоначальную поверхность. Восстановление поверхности (surface reconstruction) - это процесс соединения точек надлежащим образом для восстановления поверхности. Более формально задачу восстановления поверхности можно определить следующим образом:

Пусть дано множество точек S, полученных из первоначальной поверхности U, и необходимо построить поверхность U' так, чтобы все точки из S лежали на U'или были бы рядом с ней и поверхность U'аппроксимировала изначальную поверхность U .

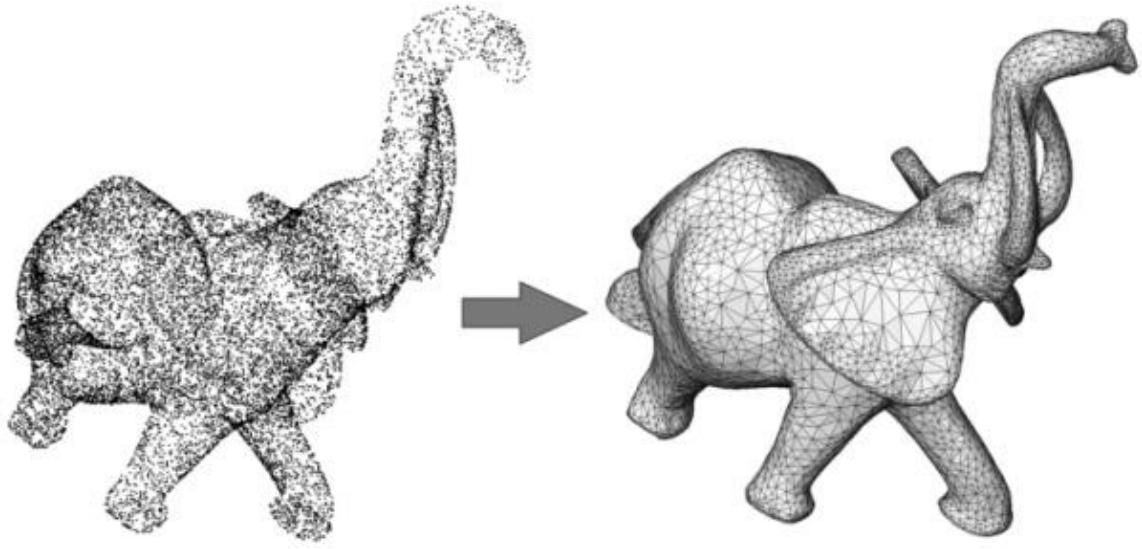


Рисунок 1.9: Пример восстановления поверхности [41].

На рисунке 1.9 показан пример восстановления поверхности: слева находится первоначальное облако точек, а справа восстановленная из облака точек поверхность.

Поверхность обычно представляется в виде треугольной или полигональной сетки (polygonal mesh). Полигональная сетка - это множество вершин, ребер и граней, которые определяют форму объекта в 3D графике и моделировании. Гранями обычно являются треугольники, четырехугольники или другие простые выпуклые многоугольники (полигоны). Треугольная сетка - это частный случай полигональной сетки, которая состоит из множество треугольников, соединенных общим ребром или общей вершиной. Полигональная сеть позволяет выразить топологические свойства поверхности и является наиболее популярным методом представления поверхности. Данное представление используется большинством алгоритмов восстановления поверхности.

Алгоритмы восстановления поверхности, в основном можно разделить на две группы: явные (explicit), которые используют вычислительную геометрию (computational geometry) и неявные (implicit).

В статье [42] рассматриваются различные алгоритмы восстановления поверхности.

Восстановленная поверхность или 3D модель может быть использована для 3D печати. В настоящее время становятся популярными и активно развиваются технологии 3D печати на основе 3D принтера. 3D принтеры используются в промышленности для изготовления промышленных деталей. Некоторые 3D принтеры могут распечатывать большинство своих деталей. В 2012 году организация Defense Distributed сообщила о планах разработать пластмассовый пистолет, который любой человек сможет скачать и напечатать на 3D-принтере. В 2013 году они закончили разработку, но вскоре после этого Государственный департамент США потребовал удалить инструкции с веб-сайта. В 2014 году начала развиваться отрасль строительства зданий на основе 3D печати. А в 2017 году компанией Air Cor был напечатан целиком жилой дом.

3D печать может осуществляться разными технологиями и с использованием различных материалов, но в основе любого способа лежит принцип послойного создания объекта печати. К технологиям, которые используются для создания слоев относятся: стереолитография, моделирование методом наплавления (fused deposition

modeling), электронно-лучевая плавка (electron-beam melting), селективное лазерное спекание (selective laser sintering) и изготовление объектов с использованием ламинации (laminated object manufacturing).

Большинство 3D принтеров для печати используют модель в файле формата STL. STL файл может содержать ошибки, такие как дыры, неправильные нормали, шумовые помехи или ошибки повторения. Для устранения этих ошибок, которые в основном возникают на стадии 3D сканирования, используется шаг восстановления STL файла. Далее при помощи специального программного обеспечения модель представляется в виде последовательности тонких слоев и STL файл преобразуется в файл формата G-Код (G-Code). G-Code - язык управления (программирования) устройств с числовым программным управлением (ЧПУ). Строка файла с G-Кодом, которая содержит одну из команд для устройства - называется кадром. 3D принтер из G-Code файла послойно печатает модель, пример 3D принтера показан на рисунке 1.10.

Рисунок 1.10: Пример 3D принтера.

1.11 ЗАКЛЮЧЕНИЕ

В данной главе рассмотрены методы получения трехмерной информации, одним из которых является стереозрение. Стереозрение - пассивный метод 3D сканиирования, который позволяет получать трехмерную информацию при помощи двух камер, не требуя специализированного дорогостоящего оборудования. Данная трехмерная информация может хранится в виде облака точек. Однако для того, чтобы использовать стерозрение для получения трехмерной модели объекта, необходимо соединить облака точек различных ракурсов объекта в одно единое облако точек. Данная задача известна, как 3D регистрация или регистрация облаков точек, которая подробна рассмотрена во второй главе. Целью данной диссертационной работы является использование стереозрения для получения 3D модели объекта.

ГЛАВА 2

РАЗРАБОТКА АЛГОРИТМОВ РЕГИСТРАЦИИ ОБЛАКОВ ТОЧЕК И СОЗДАНИЯ 3D МОДЕЛИ

В предыдущей главе было рассмотрено, как можно получить трехмерные данные и как они хранятся, в частности, в виде облаков точек. На практике для многих задач, например, для построения трехмерной модели объекта или 3D реконструкции здания, необходимо полученные облака точек соединить в одно единое облако точек. Данный процесс называется 3D регистрация (3D registration) или регистрация облаков точек. 3D регистрация - это процесс объединения нескольких облаков точек разных сцен в одно общее облако точек. Другими словами, во время регистрации множество облаков точек преобразуется в одну единую координатную систему.

На рисунке 2.1 показан пример регистрации облаков точек.

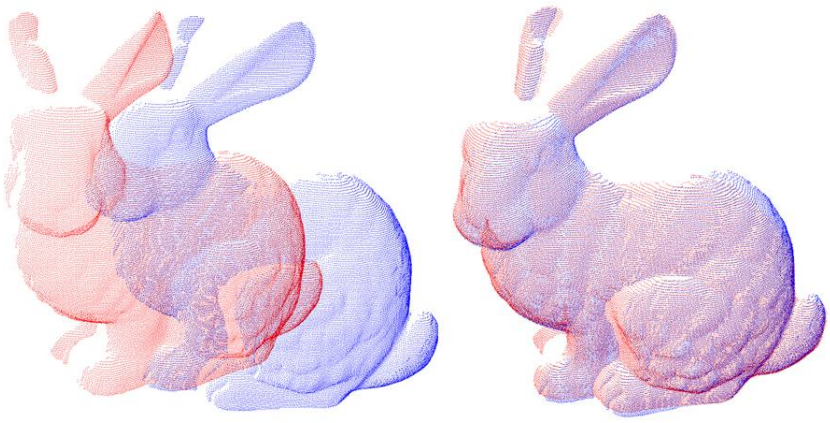


Рисунок 2.1: Пример регистрации облаков точек [43].

Алгоритмы регистрации можно разделить на следующие два типа: алгоритмы, которые используют неподвижные (жесткие, rigid) и эластичные (non-rigid) преобразования. Алгоритмы с неподвижными преобразованиями предполагают, что облака точек могут иметь только различное смещение и вращение, в то время как расстояние для каждой пары точек остается неизменным. В отличие от неподвижных алгоритмов, алгоритмы с эластичными преобразованиями способны справляться с объектами, которые меняют размер и форму в течении времени.

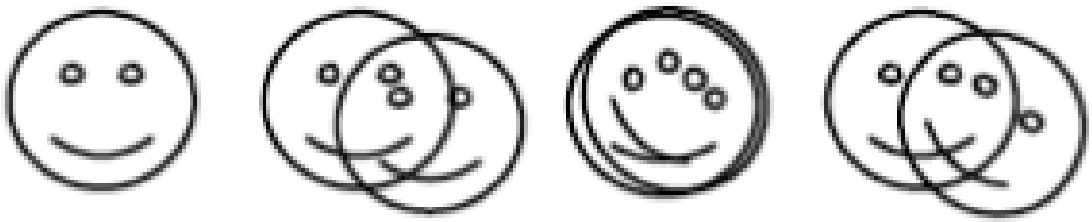


Рисунок 2.2: Объекты с неподвижными преобразованиями [44].

Рисунок 2.3: Объекты с эластичными преобразованиями [44].

На рисунках 2.2 и 2.3 приведены примеры фигур, которые можно получить при помощи неподвижных и эластичных преобразований соответственно. Как видно из рисунка 2.2 фигуры только смещены и повёрнуты относительно друг друга, а на рисунке 2.3 фигуры отличаются как по масштабу, так и по форме.

Далее будем считать, что объекты не меняют свою форму и масштаб и будем рассматривать только регистрацию с неподвижными преобразованиями.

Существует несколько подходов для решения задачи регистрации облаков точек. Ручная регистрация - пользователь вручную соединяет облака точек, полуавтоматическая - пользователь выбирает одинаковые (соответствующие) точки между двумя облаками точек и на их основе происходит регистрация, автоматическая - облака точек соединяются при помощи алгоритма регистрации. Среди автоматических алгоритмов регистрации облаков точек наиболее популярным является итеративный алгоритм ближайшей точки (Iterative Closest Point - ICP). Но этот алгоритм сильно зависит от начального приближения - для достижения хороших результатов облака точек должны находится близко друг к другу. Алгоритм ICP подробно рассмотрен в

разделе 2.4. Доказано, что алгоритм сходится к локальному минимуму [51], что в глобальном смысле не всегда является наилучшим решением. Поэтому, часто облака точек предварительно приближаются пользователем или вручную на глаз или при помощи полуавтоматического метода, когда пользователь сам определяет несколько соответствующих точек между облаками точек.

В данной главе рассматривается предложенный метод автоматической регистрации облаков точек. Идея данного метода состоит в следующем. Вначале данные подвергаются предварительной обработке: применяются фильтры для удаления шумов и выбросов (обработка облаков точек подробно рассмотрена в разделе 2.2). Далее облака точек соединяются при помощи алгоритма регистрации на основе FPFH (Fast Point Feature Histograms) дескриптора, который рассмотрен в разделе 2.3. Затем результат улучшается при помощи модификации алгоритма ICP, которая описана в разделе 2.5. Более подробно разработанный метод описан в разделе 2.6.

На основе предложенного алгоритма регистрации облаков точек и алгоритмов стереозрения, разработан метод создания трехмерной модели объекта, который описан в разделе 2.7.

2.1 ЗАДАЧА РЕГИСТРАЦИИ ОБЛАКОВ ТОЧЕК

Регистрация облаков точек - это процесс приведения нескольких облаков точек в одну общую координатную систему для дальнейшего соединения в единое облако точек. Рассмотрим более формально задачу регистрации двух облаков точек.

Пусть даны исходное (source) облако точек $(P = \{p_1, p_2, \ldots, p_n\})$ и целевое (target) облако точек $(Q = \{q_1, q_2, \ldots, q_n\})$. Необходимо найти такое неподвижное преобразование T, применяя которое к облаку точек P приведет его точки в одну общую систему координат с облаком Q.

Преобразование T из себя представляет матрицу вращения R (rotation matrix) и вектор сдвигов t (translation vector) и имеет следующий вид:

$$T = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, t = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

Применить преобразование T к точке p, означает:

$$T(p) = Rp + t,$$

где точка p имеет вид: $p = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$

Рассмотрим сдвиг и вращение в отдельности. Пусть q - точка p после смещения на t, тогда:

$$q = p + t \text{ или } \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} p_x + t_1 \\ p_y + t_2 \\ p_z + t_3 \end{pmatrix}$$

Для того, чтобы понять, как работает вращение, рассмотрим вращение объекта вдоль каждой оси. Пусть матрица R_x из себя представляет вращение точки вдоль оси X на θ радиан против часовой стрелки, тогда:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Пусть матрица R_y - это вращение точки вдоль оси Y на α радиан против часовой стрелки и имеет вид:

$$R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

Пусть матрица R_z - это вращение точки вдоль оси Z на β радиан против часовой стрелки, тогда:

$$R_z = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Матрица вращения R из себя представляет вращение по всем 3 осям, и может быть вычислена как произведение соответствующих матриц ($R_x R_y R_z$).

Алгоритмы регистрации нескольких облаков точек можно разделить на две группы: попарная (pairwise) и многовидовая (multi-view) регистрация. Попарная регистрация на каждом этапе регистрирует пару облаков точек. То есть на *i*-ом шаге *i+1*-ое облако точек приводится к координатам *i*-ого облака точек. А многовидовая регистрация пытается соединить сразу несколько облаков точек, оптимизируя результат в основном при помощи графов.

2.2 ОБРАБОТКА ОБЛАКОВ ТОЧЕК

Перед работой с облаками точек иногда требуется их предварительная обработка (например: фильтрация шумов, понижение разрешения) или вычисление дополнительной информации (например, значение нормалей). В данном разделе приводится описание этих алгоритмов.

Даунсамплинг (понижение разрешения, downsampling):

Даунсамплинг - это процесс уменьшения количества точек в облаке точек, при сохранении вида объекта или сцены, которые они представляют. Очевидно, что уменьшение количества точек снижает вычислительную стоимость и уменьшает программное время выполнения алгоритмов, работающих с облаками точек. Даунсамплинг является одним из важнейших фильтров, улучшающих эффективность действия алгоритма.

Каждую точку описывает воксель (voxel) - трехмерный куб одинакового размера для всех точек вместе со своим центроидом. Центроид - это средние координаты всех точек куба и отличается от центра куба. На рисунке 2.4 приведен пример воксельной сети облака точек:

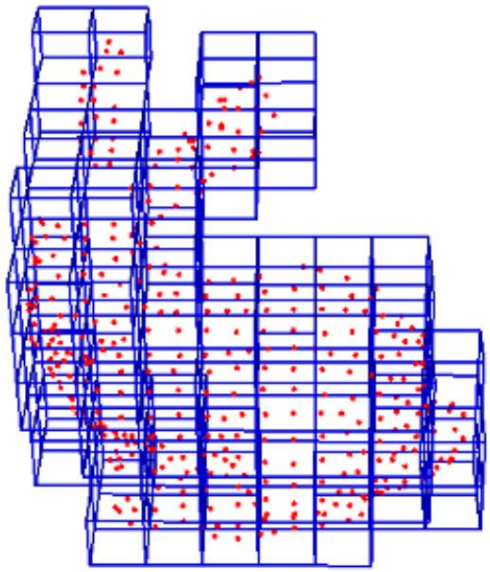


Рисунок 2.4: Воксельная сеть облака точек [45].

Существуют два типа даунсамплинга: первый заменяет все точки из вокселя центроидом вокселя - он называется воксельная сеть (voxel grid), а второй тип заменяет все точки из вокселя точкой, лежащей ближе всего к центроиду и называется равномерное распределение (uniform sampling). В первом случае новая точка добавляется к оригинальному облаку точек, а во втором случае используется существующая точка из облака точек. Метод воксельная сеть более эффективный, чем метод равномерного распределения, но зато второй метод более аккуратно описывает первоначальную поверхность.

Фильтр отсечения (pass-through filter):

Это простой фильтр, который фильтрует точки по отдельным направлениям координат и по заданному диапазону. Например, если отфильтровать облако точек по направлению оси X с заданным диапазоном $[a, b]$, то данный фильтр отбросит все точки значения координаты X которых не входят в заданный интервал.

Фильтр удаления выбросов на основе радиуса (radius outlier removal filter - ROR):

Рассмотрим метод, который лежит в основе фильтра удаления выбросов на основе радиуса. Для каждой точки рассматривается окружность заданного радиуса с центром в данной точке. Если количество точек, находящихся в этой окружности меньше заданного числа k, то данная точка отбрасывается.

На рисунке 2.5 приведен пример удаления выбросов на основе радиуса. Если $k=1$, то отбрасывается желтая точки, если же $k=2$, то зеленая точка также отбрасывается.

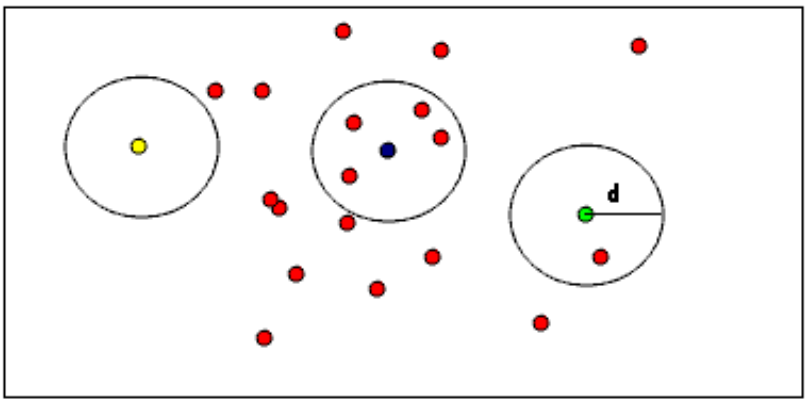


Рисунок 2.5: Пример работы фильтра удаления выбросов на основе радиуса [45].

Фильтр удаления выбросов на основе статистики (statistical outlier removal filter - SOR):

Метод, который лежит в основе фильтра выбросов на основе статистики (SOR) состоит в следующем. Вначале для каждой точки вычисляется среднее расстояние между ней и ее соседями (рассматриваются k ближайшие соседи), а затем вычисляется их среднее значение и стандартное распределение. Отбрасываются все точки, для которых среднее расстояние k ближайших соседей больше, чем M, где M определяется формулой:

$$M = avg + nSigma * \sigma,$$

где avg - среднее расстояние между всемя точками облака точек, $nSigma$ - количество повторения стандартного распределения (параметр заданный пользователем), σ - стандартное распределение.

Пример отбрасывания выбросов на основе статистики (SOR) показан на рисунке 2.6.

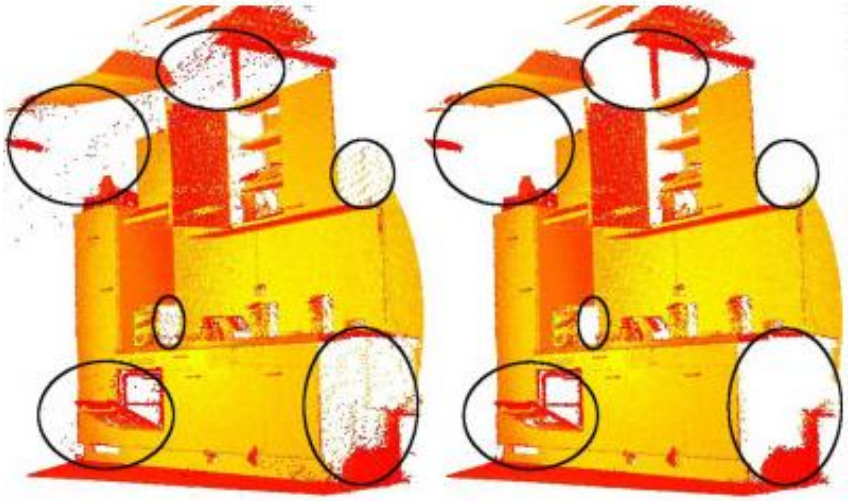


Рисунок 2.6: а) - кругами отмечены области точек, которые удаляются на основе статистики (SOR), б) - кругами отмечены эти же области с уже удаленными точками [45].

Вычисление нормалей (normal estimation):

Для многих алгоритмов работающих с облаками точек требуется информация о нормалях точек. Нормаль к поверхности в точке p из себя представляет вектор, перпендикулярный к плоскости касательной к данной поверхности в точке p (Рис. 2.7).

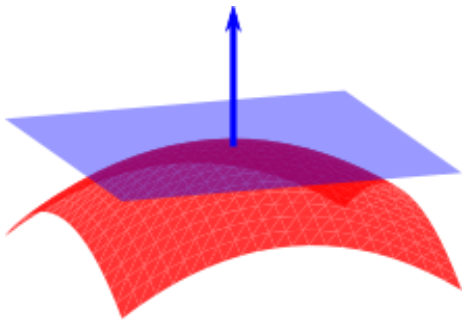


Рисунок 2.7: Нормаль к поверхности в точке p [45].

Задача вычисления нормали в точке сводится к задаче вычисления нормали к касательной плоскости, которая, в свою очередь, сводится к задаче подгонке плоскости для локальной окрестности точки. Для решения этой задачи анализируются собственные вектора и собственные значения ковариационной матрицы, построенной из ближайших соседей (k ближайших соседей, которые имеют наименьшее Евклиодовое расстояние) рассматриваемой точки:

$$C = \frac{1}{K}\sum_{i=1}^{K}(p_i - \bar{p})(p_i - \bar{p})^T,$$

где p_i - соседние точки, а $\bar{p}$ центроид точки p, то есть: $\bar{p} = \frac{1}{K}\sum_{i=1}^{K} p_i$

Не существует математического способа определения знака нормали, поэтому для этого используется метод анализа главной компоненты (Principal Component Analysis - PCA).

Точка зрения (viewpoint) - это точка откуда мы рассматриваем сцену, то есть та точка, где находится камера. Если точка зрения v_p известна, то решение будет очевидным. Для того чтобы все нормали были последовательно ориентированы относительно точки зрения v_p, они должны удовлетворять следующему уравнению:

$$\vec{n}_i(v_p - p_i) > 0$$

Пример нормалей к поверхности, ориентированных относительно точки зрения приведен на рисунке 2.8.

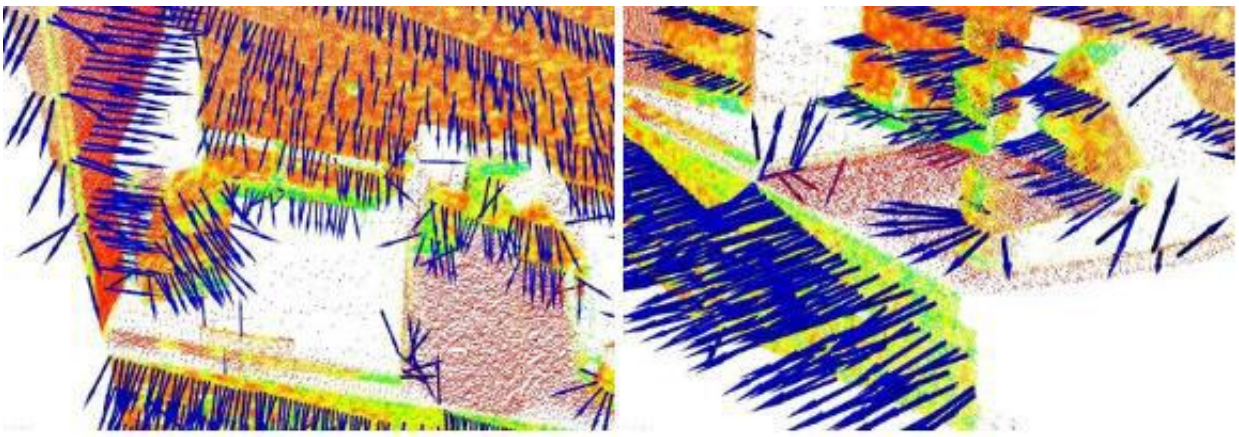


Рисунок 2.8: Пример нормалей к поверхности, ориентированных относительно точки зрения [45].

Для вычисления нормалей для облаков точек существуют два подхода:

1. Восстановить описывающую поверхность из облака точек в виде сети (меш, mesh) и вычислить нормали к поверхности по сети.
2. Используя приближения определить нормали поверхности напрямую из облака точек.

Алгоритм вычисления нормалей поверхности напрямую из облака точек, можно описать с помощью, следующей последовательности шагов:

1. Для каждой точки p из облака точек найти ближайшие ей точки.
2. Вычислить нормаль n к поверхности в точке p.
3. Проверить, чтобы нормаль n была ориентирована относительно точки зрения и повернуть ее на 180 градусов в обратном случае.

2.3 РЕГИСТРАЦИЯ НА ОСНОВЕ FPFH ДЕСКРИПТОРА

Регистрация на основе FPFH (Fast Point Feature Histograms) дескриптора является частным случаем регистрации на основе характеристик (feature-based registration). Сперва рассмотрим регистрацию на основе характеристик. Данный алгоритм является алгоритмом начальной (грубой) регистрации и его целью является уменьшение расстояние между облаками точек.

Суть алгоритма регистрации на основе характеристик состоит в следующем. Для каждой точки вычисляются дескрипторы характеристик (особенностей, feature) и на их основе находятся соответствующие точки между облаками точек. Используя данное множество соответствий вычисляется преобразование между облаками точек.

Алгоритм регистрации состоит из следующих шагов:

- вычисление ключевых точек (keypoints),
- вычисление дескрипторов характеристик,
- вычисление соответствий,
- отбрасывание плохих соответствий,
- вычисление преобразования.

Общий процесс регистрации отображен на рисунке 2.9.

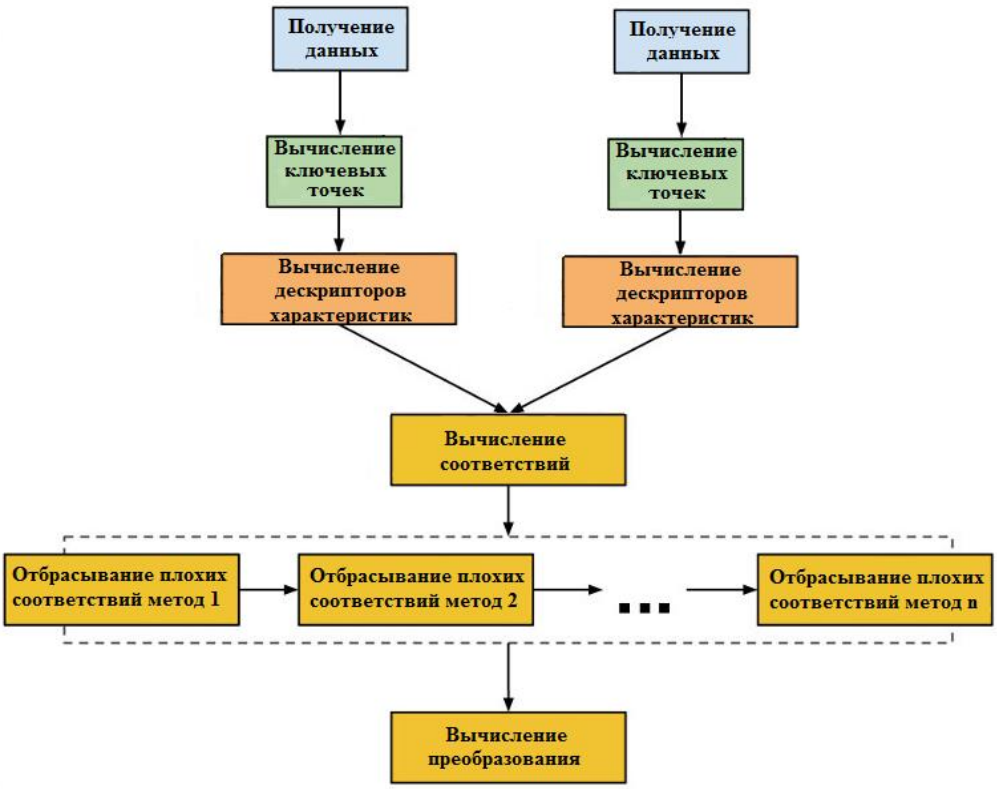


Рисунок 2.9: Процесс регистрации на основе характеристик.

Вычисление ключевых точек:

Использование всех точек при регистрации больших данных может быть не эффективным и не приемлемым в некоторых случаях. Вместо выбора всех точек может быть выбрано некоторое случайное подмножество (random selection). Но вместо выбора точек таким образом, имеет смысл выбрать точки, которые уникальны и легко различимы.

Ключевые точки (keypoints) - это специальные точки в облаке точек, которые легко найти в похожем облаке точек. Ключевые точки в литературе также встречаются, как "точки интереса" (points of interest). Другими словами, ключевые точки - это подмножество точек из облака точек, которые удовлетворяют некоторым специальным свойствам, что различает их от других точек. Ключевые точки предназначены для эффективного представления (описания) облака точек, используя намного меньше данных.

В статье [46] описано несколько типов ключевых точек и методов их извлечения. К наиболее известным и используемым типам ключевых точек относятся: SIFT, NARF, HARRIS.

Для регистрации облаков точек в данной работе используются все точки. Так как мы соединяем облака точек объекта, которые содержат небольшое количество точек (каждое облако точек содержит около 6000-9000 точек) и при вычислении ключевых точек, количество данных может быть слишком мало для правильного вычисления соотвествий.

Вычисление дескрипторов характеристик (feature descriptors):

Для поиска соответствующих точек может быть недостаточным рассматривание отдельных точек с их координатами. Для сопоставления соответствующих точек необходимо знать лежат ли точки на одинаковой поверхности или нет. Поэтому следует также рассматривать те точки, которые лежат в окрестности рассматриваемой точки.

Дескрипторы характеристик или особенностей (feature descriptors), которые в литературе также встречаются, как геометрические особенности или дескрипторы форм или просто дескрипторы или характеристики - из себя представляет структуры, которые описывают данные. Дескрипторы используют геометрическую информацию для описания данных, также иногда используется и цветовая информация. Дескрипторы бывают двух типов: локальные и глобальные. Локальные дескрипторы описывают отдельные точки, используя их локальные окрестности, а глобальные дескрипторы описывают всё облако точек или весь объект.

Хорошие дескрипторы характеристик должны быть устойчивы к шумам, быстро вычисляемы, быстро сравнимы и неизменны относительно вращения и сдвига облака точек.

В работе [46] приведено детальное описание дескрипторов характеристик, их сильные стороны и применение. PFH (Point Feature Histograms) - локальный дескриптор характеристик, который наиболее устойчив к вращению, сдвигу и плотности облаков

точек, а дескриптор FPFH (Fast Point Feature Histograms) - является ускоренной версией дескритпора PFH. Поэтому был выбран дескриптор FPFH.

Рассмотрим подробно дескриптор PFH, а потом на его основе дескриптор FPFH.

Дескриптор PFH был разработан в 2008 году Русу (Rusu) [47]. Кроме поиска соответствующих точек дескриптор также используется для классификации точек в облаке точек. Например, для нахождения краевых точек, угловых точек или точек на плоскости.

Рассмотрим вычисление дескриптора для заданной точки p. Вначале строится рамка Дарбу (Darboux [48]) для каждой пары точек внутри локальной окрестности точки p. Рамка Дарбу для пары точек p_i и p_j с нормалями n_i и n_j из себя представляет тройку чисел (u, v, w), где: $u = n_i, \quad v = (p_j - p_i) \times u, \quad w = u \times v.$

Рамка Дарбу графически изображена на рисунке 2.10.

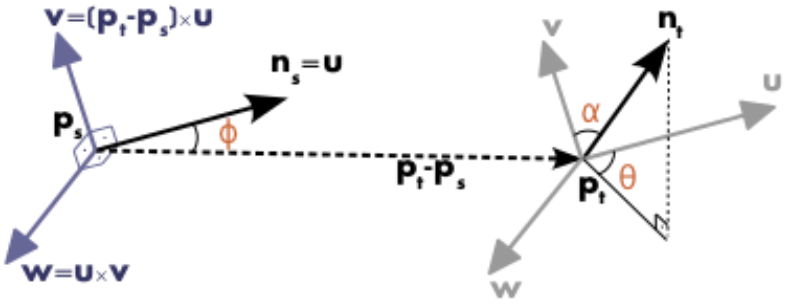


Рисунок 2.10: Графическое представление рамки Дарбу вместе с угловыми PFH характеристиками $(\alpha, \varphi, \theta)$ для точек p_s и p_t с соответствующими нормалями n_s и n_t [45].

Далее на основе рамки Дарбу вычисляются углы α, φ и θ, а также расстояние d между точками p_i и p_j:

$$d = \|p_j - p_i\|, \quad \alpha = v \cdot n_j, \quad \varphi = \frac{u(p_j - p_i)}{d}, \quad \theta = \arctan(wn_j, un_j)$$

Данная четверка записывается к гистограмме точки p. Если каждая точка имеет k соседей и в облаке точек имеется n точек, то для построения дескриптора нужно выполнить $O(nk^2)$ операций.

Вычислительная стоимость дескриптора *PFH* может быть не приемлемой для некоторых приложений. Цель дескриптора $FPFH$ оптимизировать количество операций до $O(nk)$. Дескриптор PFH для точки p вычисляет четверку параметров d, α, φ и θ для всех пар соседних точек. Определим $SPFH$ (Simplified Point Feature Histogram), который будет вычислять значения d, α, φ и θ только для пар точек, соседних с точкой p. Тогда определим дескриптор $FPFH$ для точки p, как:

$$FPFH(p) = SPFH(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_i}SPFH(p_i),$$

где w_i - расстояние от соседней точки p_i до точки p.

На рисунках 2.11 и 2.12 показана окрестность точки p_q для дескрипторов PFH и $FPFH$.

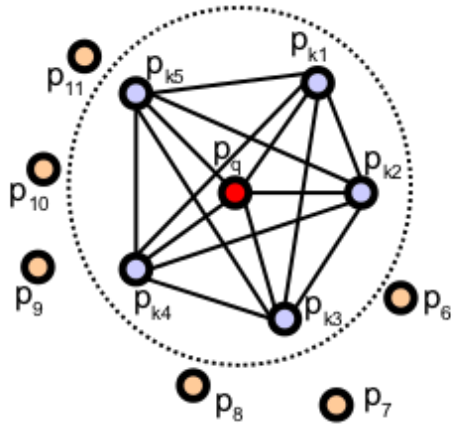


Рисунок 2.11: Окрестность точки p_q дескриптора PFH [45].

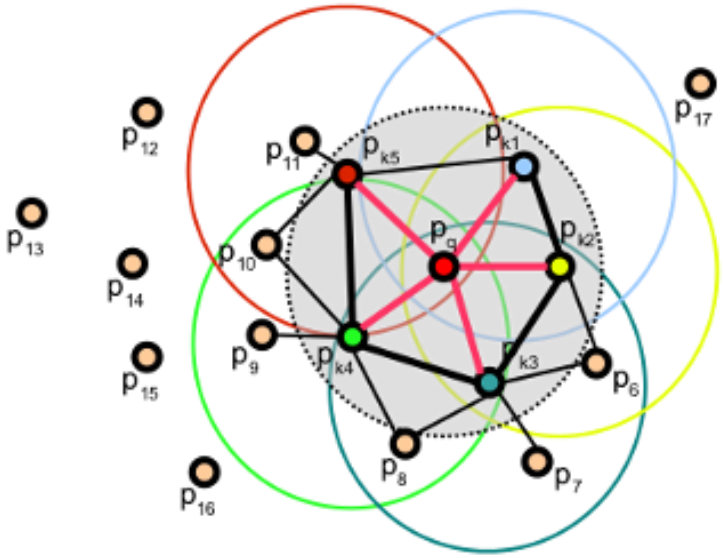


Рисунок 2.12: Окрестность точки p_q дескриптора $FPFH$ [45].

Дескриптор $FPFH$ хоть и является менее точным, чем PFH, но зато он является более эффективным и более приемлемым для приложений в реальном времени.

Вычисление соответсвий:

Соответствие - это пара точек или характеристик из двух облаков точек, которая соответствует одной и той же точке (пример показан на рисунке 2.13). Другими словами, соответствие - это пара точек, которую алгоритм нахождения соответствий отмечает, как одинаковую точку в обеих облаках точек. Соответствия могут вычисляться напрямую путем перебора, но для более эффективного поиска используются древовидные структуры данных. Нами используются *kd-деревья*.

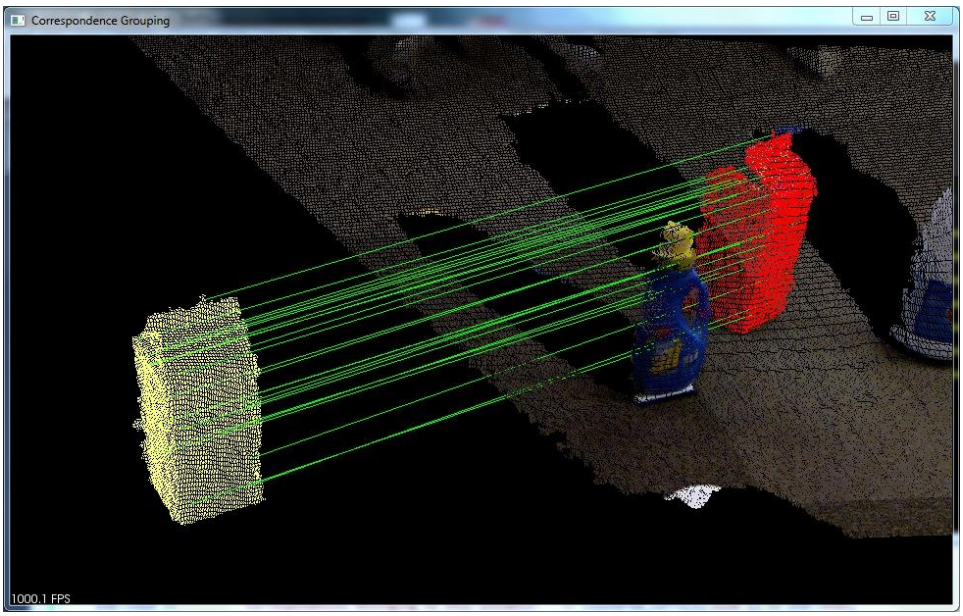


Рисунок 2.13: Соответствующие точки между облаками точек.

Отбрасывание плохих соответствий:

Из-за шума или других факторов в множестве найденных соответствий присутствуют плохие (ложные) соответствия, которые отрицательно влияют на результат регистрации. Существует множество алгоритмов для отбрасывания плохих соответствий. Один или несколько из этих алгоритмов могут быть применены. Ниже рассмотрим алгоритмы отбрасывания плохих соответствий:

1. Отбрасывание соответствий на основе расстояния. Данный метод отбрасывает пары соответствий, если расстояние между ними больше некоторого заранее заданного порогового значения.
2. Отбрасывание на основе среднего расстояния. Данный метод в отличии от предыдущего метода, вместо использования заданного фиксированного порогового значения, вычисляет и использует среднее расстояние среди всех расстояний между парами точек. Данный метод уменьшает влияние удаленных точек на процесс регистрации.

3. Отбрасывание пар с повторяющиемся соответствием. Одной точке из облака точек Q может соответствовать несколько точек из облака P, тогда из множества таких пар выбирается только одна пара с минимальным расстоянием.
4. Отбрасывание на основе метода RANSAC (Random Sample Consensus) [48]. Данный метод вычисляет преобразование для подмножества множества соответствий и применяет данное преобразование к исходному облаку точек. Далее исключаются удаленные соответствия, на основе Евклидового расстояния между точками. Этот метод эффективен для фильтрации удаленных частей (outliers).
5. Отбрасывание на основе совместимости нормалей. Данный подход использует информацию нормалей точек и отбрасывает пары с несовместимыми нормалями, то есть, если угол между нормалями больше заданного порогового значения.
6. Отбрасывание на основе границ поверхности. В некоторых случаях рассматривание соответствий, где присутствуют граничные точки поверхности может привести к ошибке и лучше отбросить такие соответствия.

На рисунке 2.14 приведены примеры алгоритмов отбрасывания, красным пунктиром отмечены те точки, которые отбрасываются из-за несоответствия условию.

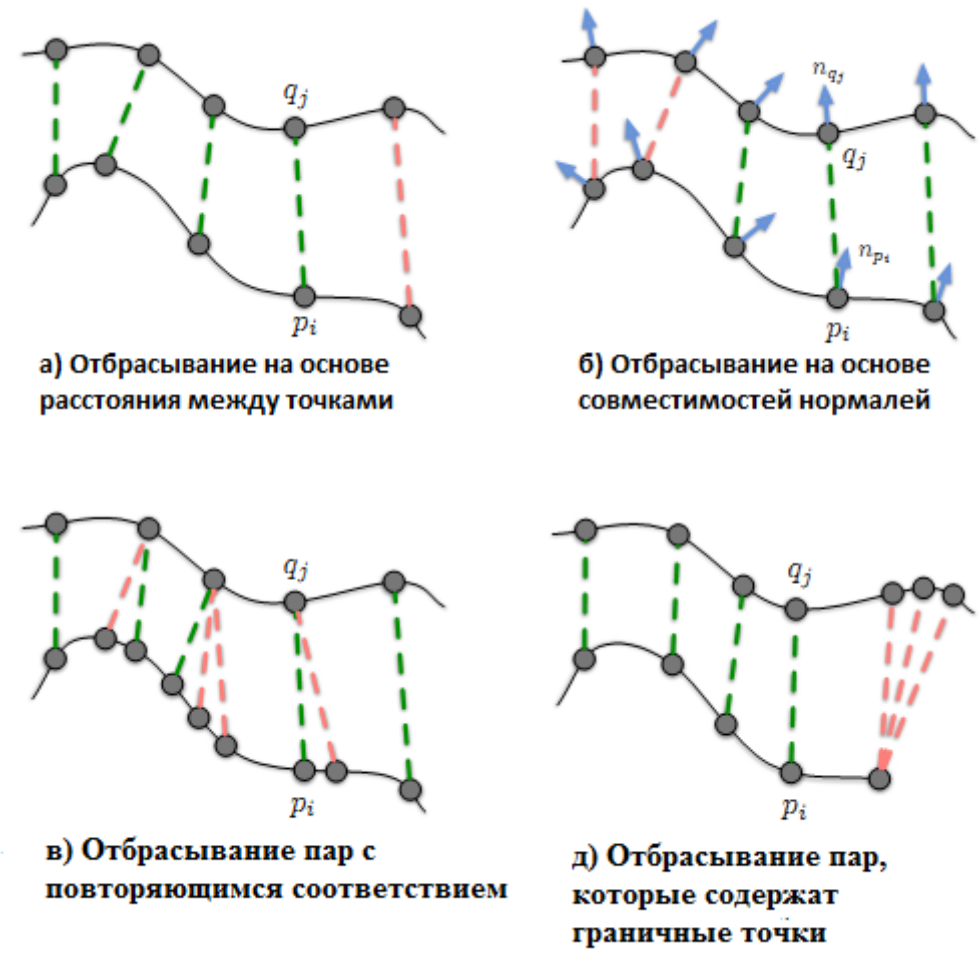


Рисунок 2.14: Примеры отбрасывания соответствий.

Вычисление преобразования:

На основе множества найденных соответствий вычисляется преобразование, которое далее применяется к исходному облаку точек для того, чтобы прививести его в одну координатную систему с целевым облаком точек. Преобразование можно вычислить при помощи метода сингулярного разложения матрицы (SVD) или метода кватерионов. Более детально методы вычисления преобразования приведены в разделе 2.4 при описании алгоритма ICP.

Обобщение алгоритма регистрации на основе FPFH дескриптора:

В данном разделе рассматривается предложенный алгоритм регистрации на основе FPFH дескриптора, который является модификацией алгоритма регистрации на основе характеристик. Алгоритм регистрации на основе характеристик состоит из следующих шагов: вычисление ключевых точек, вычисление дескрипторов характеристик, вычисление соответствий, отбрасывание плохих соответствий и вычисление преобразования.

В данном алгоритме вместо ключевых точек, берутся все точки из облака точек, так как количество точек не слишком много (для каждого облако точек около 6000-9000 точек). В качестве дескриптора характеристик используется Fast Point Feature Histograms (FPFH) [43], который является быстрой версией дескриптора PFH - этот дескриптор наиболее устойчив к вращению и сдвигам. Дескриптор FPFH вычисляет характеристики для всех точек регистрируемых облаков точек и на их основе вычисляются соответствия. Далее отбрасываются плохие соответствия на основе среднего расстояния и на основе принципа совместимости нормалей. На основе полученного множества соответствий вычисляется преобразование, которое применяется к исходному облаку точек.

2.4 ИТЕРАТИВНЫЙ АЛГОРИТМ БЛИЖАЙШИХ ТОЧЕК (ICP)

Итеративный алгоритм ближайших точек (Iterative Closest Point - ICP) - является наиболее распространенным и используемым алгоритмом для совмещения трехмерных данных (облака точек, треугольные сети, параметрические поверхности и неявные поверхности). Данный алгоритм широко используется в компьютерном зрении (computer vision). Алгоритм ICP - был разработан Беслом и Маккаем (Besl and McKay) [49] в 1992 году для регистрации трехмерных фигур, включая кривые и поверхности. Почти одновременно Ченом и Медиони (Y.Chen и G.Medioni) [50] была разработана другая версия алгоритма, которая вместо метрики точка-точка использует метрику

точка-плоскость. В 1994 году в статье [51] Жанг (Zhang) было отмечено, что формы, полученные из точек, чувствительны к шумам и изолированным частям (outliers), поэтому более надежно использовать сами точки. Он также предложил использовать *kd-деревья* для более быстрого поиска ближайших точек. Жанг является одним из первых ученых, кому принадлежит идея использовать решения на основе ICP в приложениях робототехники.

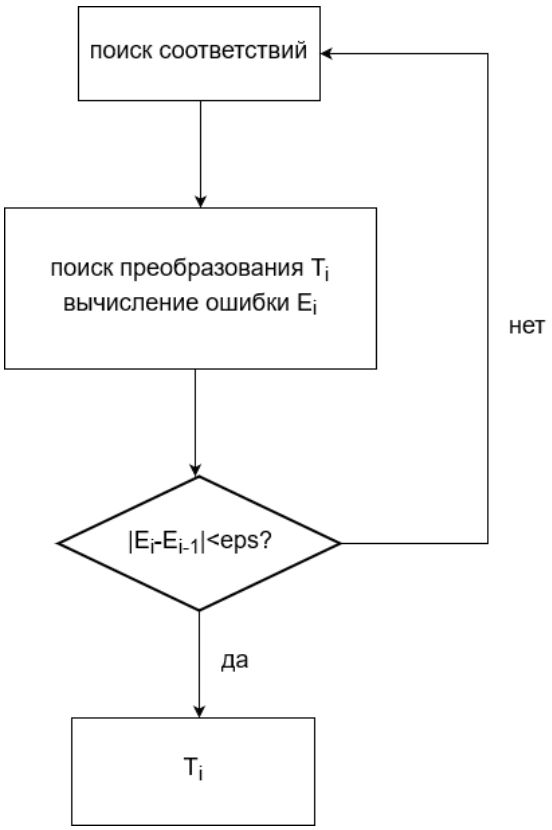
Суть алгоритма ICP заключается в том, что он итеративно соединяет облака точек. На каждой итерации уменьшается расстояние между облаками точек.

Допустим мы хотим соединить облака точек P и Q. Цель алгоритма ICP найти такое преобразование, применение которого к P минимизирует расстояние между облаками точек P и Q. Алгоритм состоит из следующих основных шагов:

1) Поиск соответствующих точек между двумя облаками точек. Соответствующие точки определяются по принципу ближайшей точки, т.е для каждой точки из облака P находится ближайшая ей точка из облака Q.
2) Поиск преобразования (трансформации, transformation), применение которого к облаку точек P, уменьшает ошибку E. Преобразование из себя представляет матрицу поворотов (rotation matrix) R и вектор сдвигов (translation vector) T.
3) Применение вычисленного преобразования к облаку точек P.

Шаги 1-3 повторяются до тех пор, пока изменение ошибки E не становится меньше некоторого заранее заданного порогового значения.

ICP алгоритм можно изобразить с помощью следующей диаграммы:



Рассмотрим эти шаги более подробно.

Поиск соответствующих точек:

Для каждой точки из облака P находится ближайшая ей точка в облаке Q и эта пара точек будет соответствующими точками. Так как облака точек могут перекрываться (совпадать) не полностью, а частично, то нужно рассматривать те точки, которые лежат в перекрывающейся области.

Пусть облако точек P имеет N_p точек, а Q имеет N_q точек, тогда в самой простой реализации путем перебора, количество операций будет $O(N_p N_q)$ или $O(N^2)$, если $N_p \approx N_q \approx N$. Для эффективного решения задачи поиска ближайших точек обычно используются древовидные структуры данных, в частности *kd-деревья*. В данном случае сложность алгоритма будет $O(N_p \log N_q)$.

Поиск преобразования и вычисление ошибки:

Необходимо найти преобразование T - то есть матрицу поворотов (rotation) R и вектор сдвигов (translation) t, которое минимизирует ошибку E. Данная ошибка показывает насколько хорош результат регистрации. Для начала рассмотрим, как вычисляется ошибка.

В первоначальной версии алгоритма [49] для вычисления ошибки E используется метрика точка-точка (point-to-point), которая определяется по формуле:

$$E = \frac{1}{N}\sum_{i=1}^{N}\left\|\left(Rp_i + t\right) - q_i\right\|^2 ,$$

где R - матрица поворотов, t - вектор сдвигов, p_i и q_i - соответствующие точки из двух облаков точек, N - количество точек, E - ошибка.

Метрика точка-точка показана на рисунке 2.15.

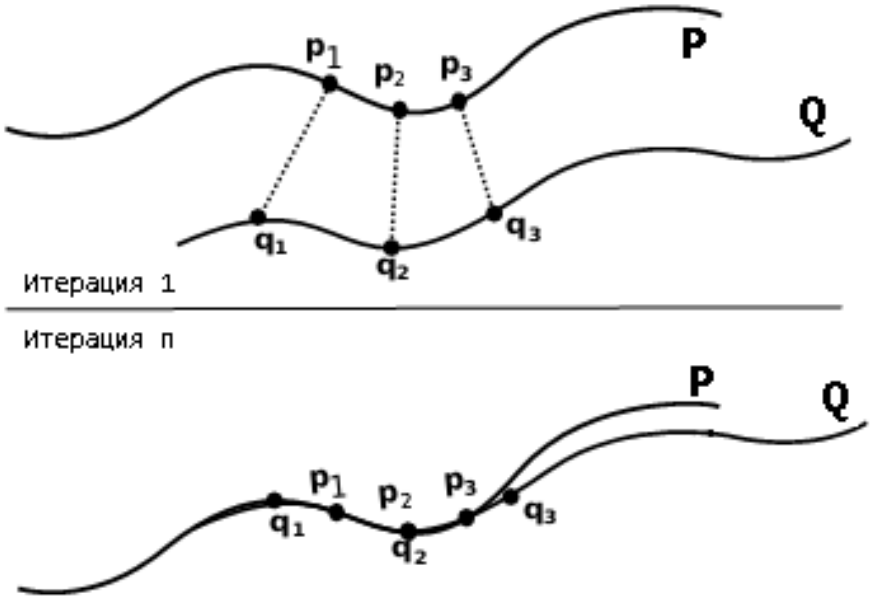


Рисунок 2.15: метрика точка-точка.

Метрика точка-точка чувствительна к выбросам (outliers) и шумам, и поэтому вместо рассмотрения непосредственно ближайшей точки (имеет наименьшее евклидовое

расстояние) q_i предпочтение следует отдавать рассмотрению ее локальной окрестности. В 1992 году Ченом и Медиони (Y.Chen и G.Medioni) в работе [50] была представлена метрика точка-плоскость (point-to-surface), которая предполагает, что облака точек локально линейны. Тогда локальная поверхность может быть определена вектором нормали n_i, которая вычисляется как наименьший собственный вектор ковариационной матрицы (covariance matrix) точек, которые окружают ближайшую точку q_i. Тогда вместо того, чтобы напрямую вычислять Евклидово расстояние между соответствующими точками можно вычислить скалярную проекцию на плоскость поверхности, определяемой вектором нормали n_i. В данном случае ошибка будет вычисляться по формуле:

$$E = \sum_{i=1}^{N} \left\| \left(\left(Rp_i + t \right) - q_i \right) n_i \right\|^2$$

Метрика точка-плоскость показана на рисунке 2.16.

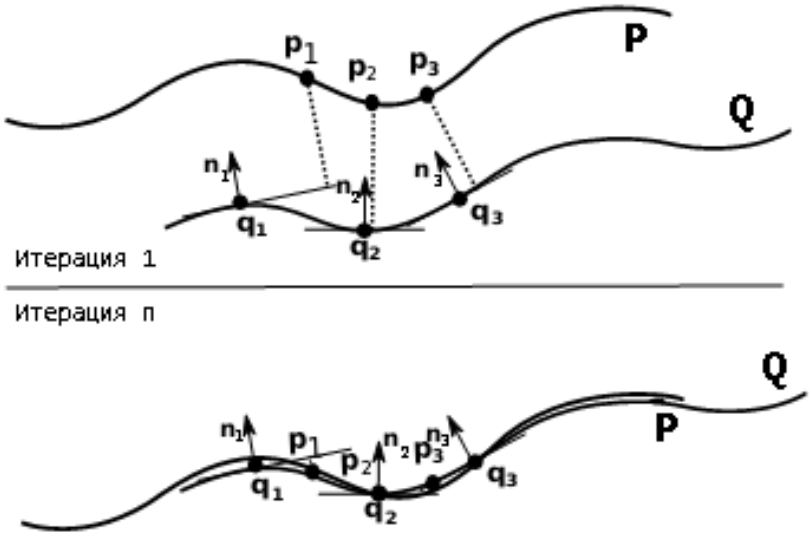


Рисунок 2.16: метрика точка-плоскость.

Итак, необходимо найти преобразование, которое минимизирует среднеквадратическую ошибку E. Для решения задачи поиска преобразования обычно используется один из следующих методов.

- Метод сингулярного разложения матрицы (SVD)
- Метод кватернионов
- Метод с использованием ортонормированных матриц
- Метод на основе вычисления двойных кватернионов

В оригинальной статье [49] предлагается использовать метод кватернионов в случае двухмерного и трехмерного пространства и метод SVD в случае, когда размерность больше трех. В статье [52] приведен сравнительный анализ этих четырех методов.

Рассмотрим ниже первые два метода более подробно.

Метод сингулярного разложения матрицы (SVD).

Для вычисления преобразования при помощи метода SVD сначала нужно вычислить центроиды (centroid) для обоих облаков точек. Центроиды вычисляются по формуле:

$$\bar{p} = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i \qquad \text{и} \qquad \bar{q} = \frac{1}{N_q}\sum_{i=1}^{N_q} q_i$$

Введем следующие определения:

$$p_i' = p_i - \bar{p} \quad \text{и} \quad q_i' = q_i - \bar{q}$$

Рассмотрим P' - множество точек p_i' и Q' - множество точек q_i' и матрицу H, которая определяется как: $H = Q'P'^T$

Далее рассмотрим сингулярное разложение матрицы H:

$$H = U\Lambda V^T$$

Тогда матрица вращения R и вектор сдвигов t будут вычислятся по формуле:

$$R = UV^T, t = \bar{q} - R\bar{p}$$

Метод кватернионов.

Вектор $\vec{q}_R = [q_0, q_1, q_2, q_3]^T$ называется единичным кватернионом, где $q_0 \geq 0$ и $q_1 + q_2 + q_3 + q_4 = 1$.

Как и в предыдущем случае вычисляются центроиды, и матрица H:

$$H = Q'P'^T$$

Далее вычисляется несимметричная матрица A по формуле: $A_{ij} = (H - H^T)_{ij}$ и на ее основе вычисляется вектор $\Delta = [A_{23} \quad A_{31} \quad A_{12}]^T$. Данный вектор используется для составления симметричной матрицы *Q(H)* размером 4 на 4:

$$Q(H) = \begin{pmatrix} tr(H) & \Delta^T \\ \Delta & H + H^T - tr(H)I_3 \end{pmatrix},$$

где *tr(H)* - это след матрицы H, то есть сумма диагональных элементов данной матрицы, а I_3 - единичная матрица размером 3 на 3.

Далее берется собственный вектор $\vec{q}_R = [q_0, q_1, q_2, q_3]^T$, соответствующий максимальному собственному значению матрицы *Q(H)* и на его основе вычисляется вращение R, как:

$$R = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{pmatrix}$$

Вектор сдвигов вычисляется, как и в предыдущем случае по формуле:

$$t = \bar{q} - R\bar{p}$$

Применение преобразования:

Вычисленное преобразование далее применяется к облаку точек P и проверяется, чтобы изменение ошибки было меньше некоторого заранее заданного порогового значения. Если условие выполняется, то алгоритм останавливается и найденное преобразование считается оптимальным решением, в противном случае алгоритм повторяется начиная с первого шага. В реализациях алгоритма существуют также другие критерии завершения алгоритма, например, максимальное количество итераций.

2.5 МОДИФИКАЦИЯ АЛГОРИТМА ICP

Более чем за 20 лет было разработано множество версий алгоритма ICP, но все они в основном влияют на один из шести нижеуказанных шагов:

1. Выбор некоторого множества точек, в одном или в обоих облаках точек. Это может быть, как само облако точек, так и его некоторое подмножество или ключевые точки.
2. Поиск соответствий между выбранными точками из одного облака точек в другом.
3. Определение веса для соответствующих точек.
4. Отбрасывание плохих соответствий. Например, отбрасывание ближайших точек, расстояние между которыми больше некоторого заданного порогового значения.
5. Определение метрики ошибки на основе пар соответствующих точек. Как было сказано выше метрикой может быть метрика точка-точка или точка-плоскость.
6. Решение задачи минимизации среднеквадратической ошибки.

Более детально разновидности ICP описаны в работе [53].

На основе экспериментальных данных была выбрана следующая версия алгоритма ICP, которая дает наилучший результат регистрации на наших входных данных. Примеры

некоторых экспериментов и сравнение модификаций ICP рассмотрены в разделе 3.8, когда описывается реализация модуля регистрации.

В качестве выбора точек используются все точки из обоих облаков точек. Если данные имеют большой объем (количество точек облака больше 100000, так как сложность алгоритма равна $O(N_p \log N_q)$), то предлагается применить даунсамплинг к облакам точек.

Для вычисления соответствий используются *kd-деревья*, а соответствиям присваивается одинаковый вес.

Для отбрасывания соответствий предлагается использовать несколько принципов. Вначале, отбрасываются соответствия на основе среднего расстояния. Далее отбрасываются соответствия на основе совмещения нормалей, то есть отбрасываются те соответствия, угол между нормалями которых больше некоторого значения.

Предлагается в качестве метрики ошибки использовать метрику точка-плоскость, а для вычисления преобразования для метрики точка-плоскость использовать линейную оптимизацию наименьших квадратов [54]. Идея заключается в следующем:

Для малых углов $\alpha \approx 0$ используя приближение $\sin\alpha \approx 0$ и $\cos\alpha \approx 1$ аппроксимируем матрицу поворотов R матрицей R, которая имеет вид:

$$R \approx R' = \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix},$$

где α, β, γ - углы вращения относительно осей X, Y, Z соответственно.

При минимизации функции ошибки вместо матрицы R используем R':

$$E = \sum_{i=1}^{N} \left\| \left(\left(R' p_i + t \right) - q_i \right) n_i \right\|^2 \qquad (2.5.1)$$

Рассмотрим слагаемое $\left(\left(R' p_i + t \right) - q_i \right) \cdot n_i$, которое может быть записано, как линейное выражение, зависящее от шести параметров $\alpha, \beta, \gamma, t_x, t_y$ и t_z:

$$\left(\left(R'p_i + t\right) - q_i\right)n_i = \left(R'\begin{pmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} - \begin{pmatrix} q_{ix} \\ q_{iy} \\ q_{iz} \end{pmatrix}\right)\begin{pmatrix} n_{ix} \\ n_{iy} \\ n_{iz} \end{pmatrix} =$$

$$= \left[\left(n_{iz}p_{iy} - n_{iy}p_{iz}\right)\alpha + \left(n_{ix}p_{iz} - n_{iz}p_{ix}\right)\beta + \left(n_{iy}p_{ix} - n_{ix}p_{iy}\right)\gamma + n_{ix}t_x + n_{iy}t_y + n_{iz}t_z\right] -$$

$$\left[n_{ix}q_{ix} + n_{iy}q_{iy} + n_{ixz}q_{iz} - n_{ix}p_{ix} - n_{iy}p_{iy} - n_{iz}p_{iz}\right]$$

Для всех i, $1 \le i \le N$, выражение $\left(\left(R'p_i + t\right) - q_i\right) \cdot n_i$ можем записать в виде следующего матричного выражения:

$$Ax - b \qquad (2.5.2),$$

где $b = \begin{pmatrix} n_{1x}q_{1x} + n_{1y}q_{1y} + n_{1z}q_{1z} - n_{1x}p_{1x} - n_{1y}p_{1y} - n_{1z}p_{1z} \\ n_{2x}q_{2x} + n_{2y}q_{2y} + n_{2z}q_{2z} - n_{2x}p_{2x} - n_{2y}p_{2y} - n_{2z}p_{2z} \\ \vdots \\ n_{Nx}q_{Nx} + n_{Ny}q_{Ny} + n_{Nz}q_{Nz} - n_{Nx}p_{Nx} - n_{Ny}p_{Ny} - n_{Nz}p_{Nz} \end{pmatrix}$,

$x = \begin{pmatrix} \alpha & \beta & \gamma & t_x & t_y & t_z \end{pmatrix}^T$ и

$A = \begin{pmatrix} a_{11} & a_{12} & a_{12} & n_{1x} & n_{1y} & n_{1z} \\ a_{21} & a_{22} & a_{23} & n_{2x} & n_{2y} & n_{2z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & n_{Nx} & n_{Ny} & n_{Nz} \end{pmatrix}$,

$a_{i1} = n_{iz}p_{iy} - n_{iy}p_{iz}$, $a_{i2} = n_{ix}p_{iz} - n_{iz}p_{ix}$, $a_{i3} = n_{iy}p_{ix} - n_{ix}p_{iy}$.

Таким образом, вместо минимизации уравнения (2.5.1) будем минимизировать (2.5.2):

$\min\limits_{R,t} \sum\limits_{i=1}^{N} \left\|\left(\left(R'p_i + t\right) - q_i\right)n_i\right\|^2 = \min\limits_{x} \left|Ax - b\right|^2$, которое является стандартной линейной задачей наименьших квадратов и может быть решена при помощи метода SVD.

2.6 РАЗРАБОТАННЫЙ МЕТОД РЕГИСТРАЦИИ ОБЛАКОВ ТОЧЕК

На основе вышеуказанных алгоритмов предложен следующий метод автоматической регистрации облаков точек. Идея данного метода изображена на рисунке 2.17.

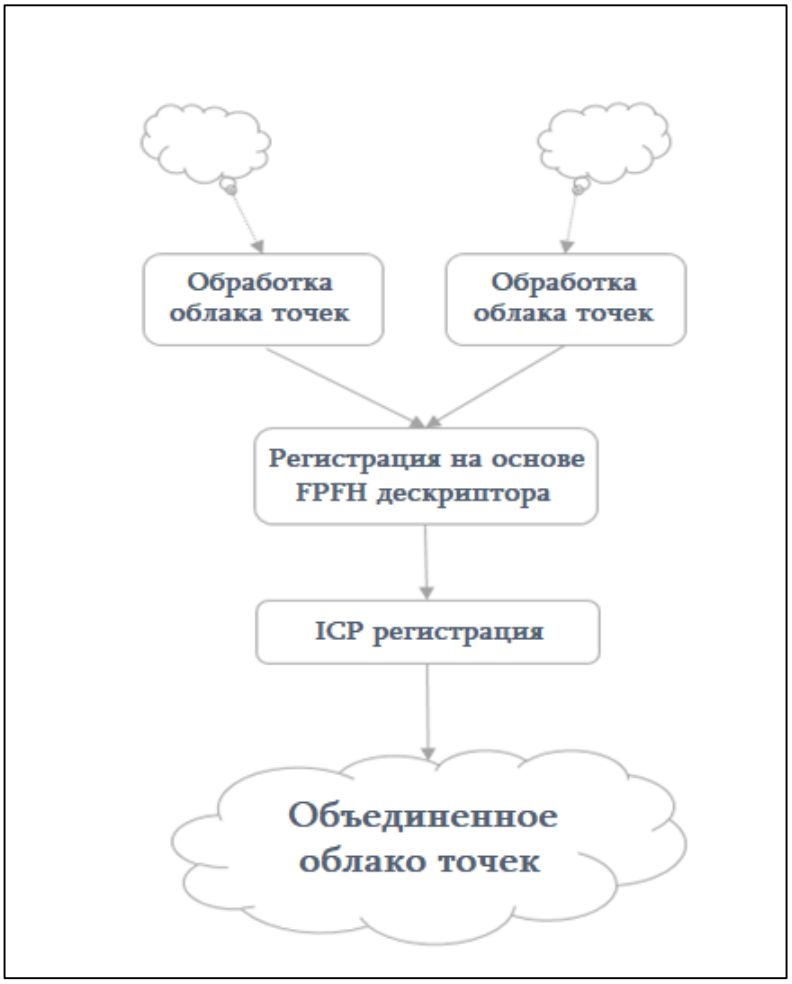


Рисунок 2.17: Разработанный алгоритм регистрации.

Как видно из рисунка метод состоит из следующих шагов:

1) Обработка облаков точек - удаляются выбросы (outliers, изолированные точки) при помощи фильтра SOR (Statistical Outlier Removal), затем вычисляются нормали облака точек и при необходимости можно применить алгоритм понижения разрешения облака точек (даунсамплинг). В нашем случае даунсамплинг не применяется, так как количество точек каждого облако точек мало (около 6000-9000 точек).

2) В качестве начальной или грубой регистрации применяется рассмотренный в разделе 2.3 алгоритм регистрации на основе FPFH дескриптора.
3) Далее применяется алгоритм регистрации ICP (рассмотренная в разделе 2.5 модификация), который также возвращает значение ошибки. Регистрацию в основном можно считать хорошей, если значение ошибки меньше 0.1.

Приведем сравнение разработанного метода регистрации облаков точек со стандартным ICP на примере данных Стэндфордского кролика [49]. Облака точек повернуты по отношению друг к другу на 45 гардусов, эти облака точек показаны на рисунке 2.18. На рисунке 2.19 а) - показана регистрация только при помощи стандартного алгоритма ICP: как видно, плохой результат, потому что расстояние между облаками точек большое. А на рисунке 2.19 б) - показана регистрация при помощи разработанного метода регистрации.

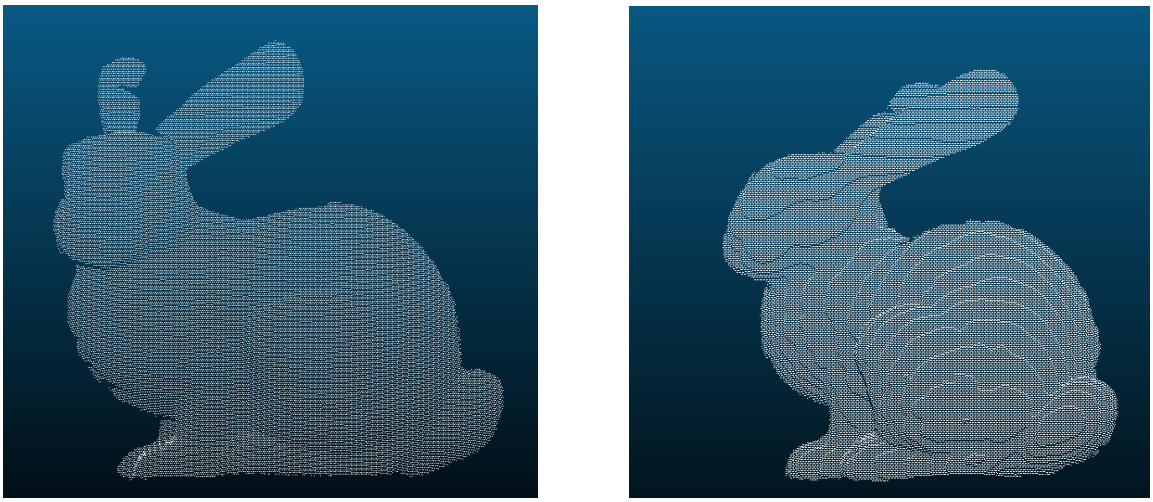
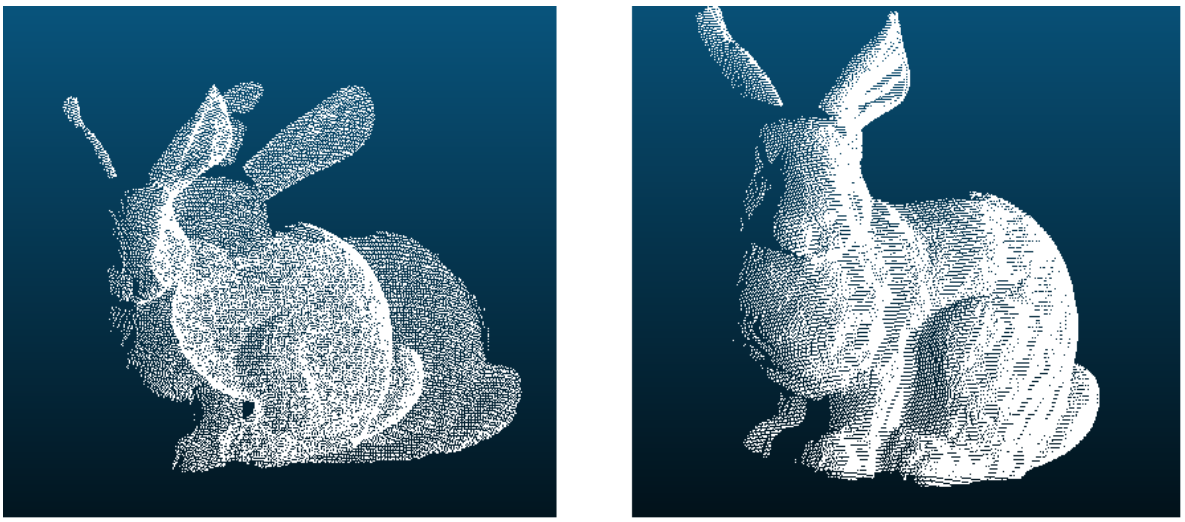


Рисунок 2.18: Регистрируемые облака точек.



а) б)

Рисунок 2.19: а) Регистрация только при помощи ICP, б) Разработанный метод регистрации.

Результаты разработанного алгоритма на реальных данных описаны в разделе 3.8. Данный метод в отличии от ICP не нуждается в предварительном приближении облаков точек и более устойчив к наличию шума.

2.7 РАЗРАБОТАННЫЙ МЕТОД СОЗДАНИЯ 3D МОДЕЛИ ОБЪЕКТА

На основе обобщения, рассмотренных в главе 1 алгоритмов стереозрения и предложенного алгоритма регистрации облаков точек, предлагается следующий метод создания 3D модели реального объекта.

Предлагаемый метод состоит из следующих шагов:

1) Взять две веб-камеры, зафиксировать их по отношению друг друга и откалибровать. Таким образом, вычисляются внутренние параметры камер и их геометрическое расположение по отношению друг другу.
2) При помощи этих камер сфотографировать объект с различных ракурсов. Таким образом, получаем последовательность стереоизображений, где каждому стереоизображению соответствует один ракурс объекта.

3) Для каждой стереопары изображения вычисляется карта смещений при помощи алгоритма полуглобального сопоставления (SGM).
4) Для каждой карты смещений при помощи триангуляции вычисляется карта глубины и преобразуется в облако точек.
5) Для каждого полученного облака точек выделяется объект от сцены, чтобы получить облако точек только объекта. Таким образом, получаем последовательность облаков точек различных ракурсов объекта.
6) Полученную последовательность облаков точек соединяем в одно единое облако точек при помощи разработанного алгоритма 3D регистрации.

После регистрации всех облаков точек получаем облако точек, которое соответствует полной модели объекта. По данному облаку точек можно восстановить первоначальную поверхность, преобразовав его в прямоугольную сеть. Данный формат более удобен для множества приложений, в частности, для 3D печати.

Более подробная реализация данного метода вместе с экспериментальными результатами рассмотрены в главе 3.

2.8 ЗАКЛЮЧЕНИЕ

В данной главе рассмотрена проблема регистрации трехмерных данных. Многие алгоритмы 3D регистрации требуют вмешательства пользователя или хорошего начального приближения. Наиболее популярным алгоритмом регистрации облаков точек является итеративный алгоритм ближайшей точки (ICP). Но данный алгоритм имеет известную проблему локального минимума. Для достижения хороших результатов при использовании алгоритма ICP, необходимо, чтобы расстояние между облаками точек было маленьким.

Для решения этой задачи в данной главе рассмотрен предложенный новый метод регистрации облаков точек. Сущность данного метода состоит в следующем:

- Вначале облака точек подвергаются предварительной обработке и применяется фильтр удаления выбросов (изолированных точек) на основе статистики (SOR). Такой подход позволяет исключить отрицательное влияние изолированных точек на результат регистрации. Также вычисляются нормали облаков точек и при необходимости применяется даунсамплинг.
- После этого в качестве начального приближения облаков точек применяется предложенный в работе алгоритм регистрации на основе FPFH дескриптора. Применение этого алгоритма приводит к уменьшению расстояния между облаками точек.
- В конце полученный результат улучшается при помощи предложенной модификации алгоритма ICP, который в качестве метрики измерения ошибки использует метрику точка-плоскость.

Далее рассматривается предложенный метод генерации 3D модели реального объекта, который при помощи алгоритмов стереозрения получает облака точек различных ракурсов и далее соединяет эти облака при помощи разработанного алгоритма регистрации.

Предложенный метод построения трехмерной модели был реализован с помощью разработанного программного комплекса. Данный программный комплекс описан в главе 3. Он позволяет получить трехмерную модель реального объекта используя две обычные веб камеры.

ГЛАВА 3

ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО КОМПЛЕКСА И ЭКСПЕРИМЕНТАЛЬНЫЕ ДАННЫЕ

В данной главе описан реализованный программный комплекс, который комбинируя теоретические основы стереозрения с алгоритмами 3D регистрации и алгоритмами, работающими с трехмерными данными, позволяет получить трехмерную модель реального физического объекта. В главе также приведены результаты экспериментов. В ходе диссертационной работы было проведено множество экспериментов. В описание данной главы вошли примеры основных экспериментов. Работа функций программного комплекса в основом показана на примере объекта кассы медведя.

Для получения трехмерной модели объекта достаточно иметь две обычные веб-камеры. Необходимо знать их параметры и геометрическое расположение по отношению друг к другу. Данные параметры можно вычислить, если предварительно откалибровать камеры. Объект, для которого строится модель, фотографируется с разных ракурсов одновременно двумя камерами. Далее, на основе этой последовательности стереоизображений, разработанный программный комплекс строит трёхмерную модель.

В разделе 3.1 описаны используемые технологии, библиотеки, программные и аппаратные обеспечения, которые использовались при реализации и тестирования. В разделе 3.2 описаны детали реализации, модули программы и экспериментальные данные. С 3.3 по 3.10 раздел подробно описываются основные модули программы и их функции вместе с экспериментами. В разделе 3.11 описано применение полученной трехмерной модели, которая была распечатана при помощи 3D принтера.

3.1 ИСПОЛЬЗУЕМЫЕ ПРОГРАММНЫЕ И АППАРАТНЫЕ СРЕДСТВА

Программные средства: Программное обеспечение разработано на языке C++ в среде разработки Microsoft Visual Studio с использованием библиотек OpenCV и PCL.

OpenCV (Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) [55], [56] - крупнейшая библиотека для работы с изображениями, которая содержит множество алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения. Может быть использована для языков C, C++, Java и Python, поддерживает параллельную реализацию алгоритмов на GPU. Нами используются следующие модули библиотеки OpenCV:

- opencv_core - модуль основной функциональности. Включает в себя базовые структуры, вычисления (математические функции, генераторы случайных чисел) и линейную алгебру.
- opencv_imgproc - содержит функции обработки изображений (фильтры, геометрические преобразовния и т.д.).
- opencv_highgui - используется для ввода/вывода изображений, создания полбзовательского интерфейса.
- opencv_calib3d - используется для калибровки камер, поиска стерео соответсвий и обработки трехмерных данных.

Point Cloud Library (PCL) [45] - библиотека для работы с 2D/3D изображениями и трехмерными данными, который в основном хранятся в виде облаков точек. Содержит множество алгоритмов для визуализации, фильтрации и соединения облаков точек.

PCL - библиотека с открытым исходным кодом, выпущена под BSD лицензией и может быть свободно использована в коммерческих и исследовательских целях. Разработкой библиотеки занимаются множество ученных и разработчиков из различных организаций и институтов по всему миру. Проект финансово поддерживается многими организациями, к которым относятся: Open Perception, Willow Garage, NVidia, Google,

Toyota, Trimble, Urban Robotics, Honda Research Institute и другие. PCL - мультиплатформенная библиотека реализованная на языке C++ и работает для Windows, Linux, MacOS и Androd/iOS. Библиотека PCL состоит из различных модулей, которые могут быть скомпилированы и использованы отдельно, что важно для систем с ограниченной памятью. Модули PCL изображены на рисунке 3.1.

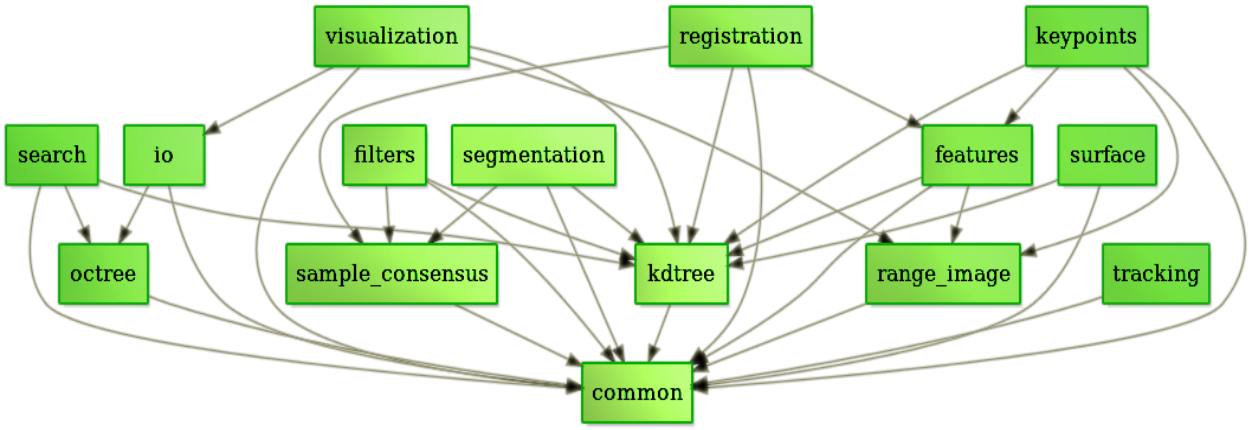


Рисунок 3.1: Модули библиотеки PCL [45].

PCL зависит и использует библиотеки Boost, Eigen, FLANN, OpenNI, Qhull, VTK.

Boost - это набор библиотек классов для C++, которые охватывают такие отрасли, как: алгоритмы, структура данных, многопоточное программирование, графы, линейная алгебра, итераторы и т. д.

Eigen - библиотека линейной алгебры для C++ с открытым исходным кодом.

FLANN - библиотека, которая выполняет быстрый поиск ближайших соседей в многомерном пространстве.

OpenNI (Open Natural Interaction) - SDK с открытым исходным кодом и используется для работы с устройствами (3D сенснорами) на основе OpenNI.

Qhull - библиотека, которая используется для вычисления выпуклой оболочки, триангуляции Делоне, диаграммы Вороного.

VTK - библиотека на C++, используется в трехмерной графике для визуализации и рендеринга (rendering) трехмерных данных.

CMake - утилита для автоматизации сборки программного обеспечения из исходного кода. CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt. Нами использовался CMake для генерации dll (dynamic link library) библиотек из исходного кода.

Аппаратные средства: Для экспериментов нами использовались две веб-камеры модели Logitech HD Pro Webcam C920 [57]. Logitech HD Pro Webcam C920 - позволяет записывать видео в формате Full HD 1080p (до 1920 x 1080 пикселей). Камера имеет два встроенных стереомикрофона с автоматическим шумоподавлением и автоматическую коррекцию низкого уровня освещенности. Камеры закреплены параллельно друг к другу на фиксированном расстоянии (приблизительно 13 сантиметров).

Рисунок 3.2: Logitech C920 веб-камеры.

Также была протестирована Minoru 3D Webcam [14] стереокамера. Minoru 3D из себя представляет две встроенные камеры с разрешениями 640x480 пикселей и одним USB выходом. Расстояние между камерами 6 сантиметров, а максимальная частота кадров для каждой камеры 30 fps (frames per second). Из-за того, что данная стереокамера

имеет один USB выход для обоих камер, технически возникли проблемы с получением изображений размером 640x480 пикселей одновременно для двух камер под Windows. Максимальное разрешение, которое можно получить для снимков, снятых одновременно двумя камерами для Windows составляет 480x360 пикселей. Поэтому, для получения 640x480 изображений был использована операционная система Linux.

Рисунок 3.3: Minoru 3D веб-камера.

3.2 ОПИСАНИЕ РЕАЛИЗАЦИИ ПРОГРАММНОГО КОМПЛЕКСА

В рамках данной диссертационной работы был разработан программный комплекс, который позволяет получать трехмерную модель реального физического объекта при помощи двух веб камер. 3D модель может быть далее использованна для 3D печати или применятся в других приложениях трехмерного моделирования.

Опишем процесс получения трехмерной модели при помощи двух обычных веб-камер. Необходимо зафиксировать камеры по отношению друг к другу (на расстоянии приблизительно 15 сантиметров) так, чтобы они были максимально параллельны друг к другу, как показано на рисунке 3.4.

Рисунок 3.4: Зафиксированные веб-камеры.

Далее при помощи модуля калибровки (Calibration) следует откалибровать камеры (т.е. получить внешние и внутренние параметры камер). Процесс и модуль калибровки более подробно описаны в разделе 3.2.1. Процесс калибровки может быть выполнен несколько раз, пока не будут получены хорошие результаты. После удачной калибровки необходимые параметры камер сохраняются в калибровочном файле. Калибровку камер достаточно выполнить один раз для данного фиксированного положения (конфигурации) камер. Если геометрическое расположение камер по отношению друг к другу изменится, тогда необходимо повторить процесс калибровки.

Далее берется объект, трехмерную модель которого мы хотим получить, и размещается перед камерами, как показано на рисунке 3.5. Если расстояние между камерами около 15 сантиметров, то объект лучше всего расположить приблизительно на расстоянии 40-50 сантиметров от камер. Затем, вращая объект, следует сфотографировать каждый его ракурс одновременно двумя камерами. Два последовательных ракурса объекта должны пересекаться хотя бы на 50-60%. Чем меньше угол вращения, тем больше вероятность, что получится хороший результат.

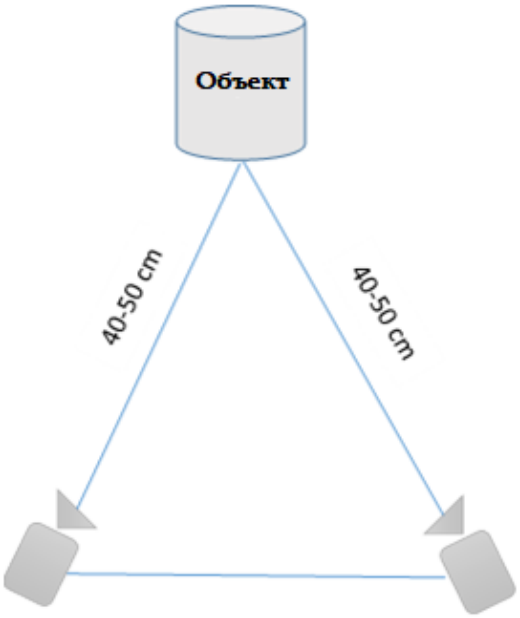


Рисунок 3.5: Пример получения стереопары изображений объекта.

Таким образом, получаем последовательность стереопар (изображения левой и правой камеры). Полученную последовательность стереоизображений необходимо передать нашей программе для получения 3D модели. В экспериментах для построения трехмерной модели мы использовали от 15 до 25 пар изображений.

Программа работает следующим образом:

Каждая стереопара изображений ректифицируется и на основе параметров камер, которые записаны в калибровочном файле, вычисляется карта смещений при помощи алгоритма стереосоответствия SGM (алгоритм SGM подробно описан в разделе 1.7). Далее по карте смещений вычисляется карта глубины и преобразуется в облако точек. Для каждого облака точек объект отделяется от сцены и записывается в соответствующий файл. Таким, образом для каждого вращения объекта получается соответствующее ему облако точек. Далее полученную последовательность облаков точек следует соединить в одну единую модель. Данный процесс выполняется при помощи алгоритма регистрации облаков точек, который описан в главе 2. Если расстояние между облаками точек небольшое, то непосредственно применяется алгоритм ICP. В противном случае, сначала применяется алгоритм регистрации на

основе FPFH дескриптора в качестве начального приближения облаков точек и лишь затем применятся алгоритм ICP. В результате получается облако точек, которое соответствует полной 360 градусной модели объекта. Данное облако точек при помощи алгоритмов восстановления поверхности преобразуется в полигональную сеть и записывается в файл формата STL. Полученная полигональная сеть является 3D моделью объекта, которая может быть использована в различных целях. В частности, быть передана 3D принтеру для печати.

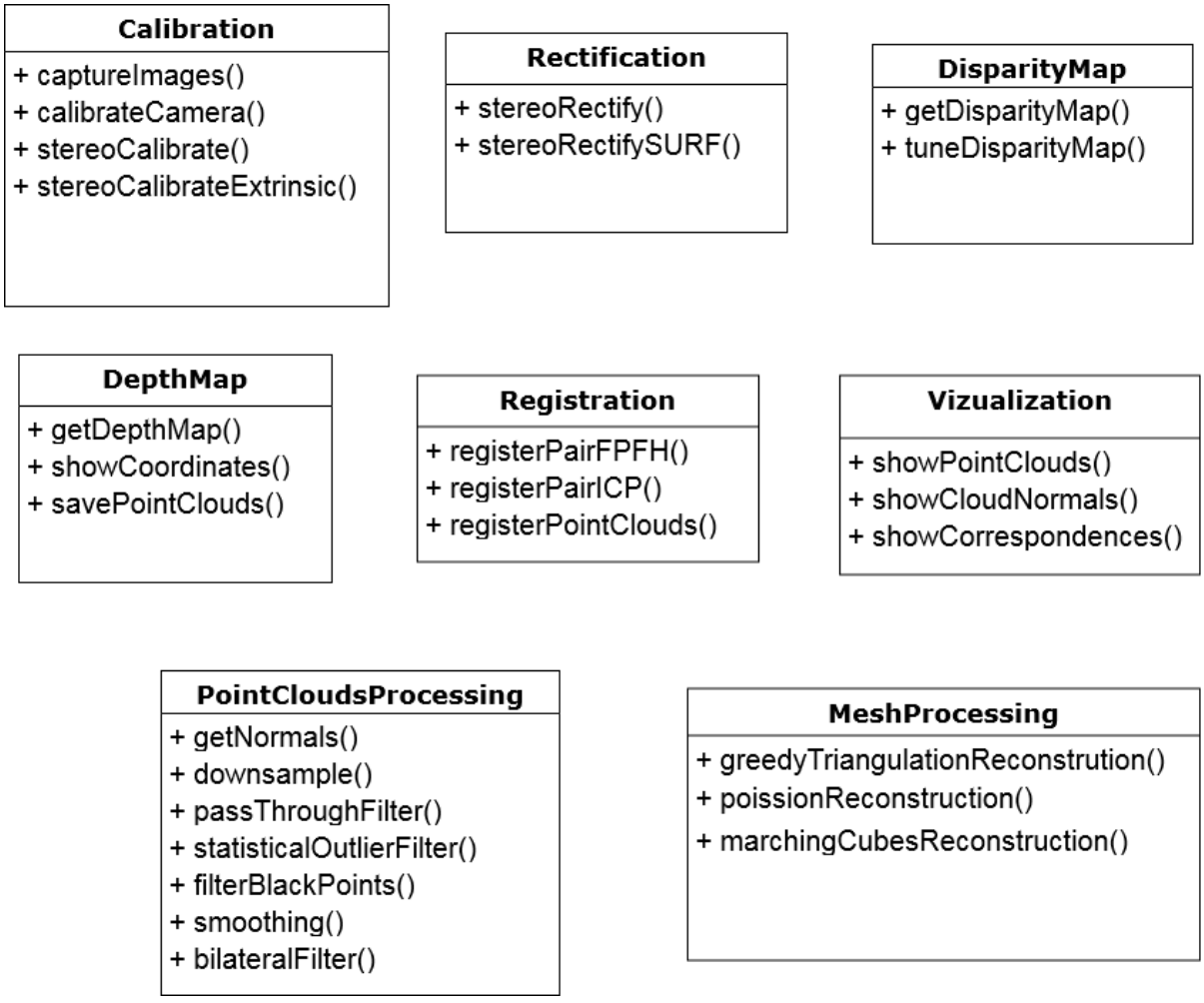


Рисунок 3.6: Основные модули программы с основными функциями.

Программный комплекс состоит из различных модулей (классов, частей), каждый из которых может быть использован отдельно. Например, модуль стерео соответствия (DisparityMap) - для вычисления карты смещений по стереопаре изображений, модуль калибровки (Calibration) для калибровки камер, модуль регистрации (Registration) для совмещения имеющихся облаков точек. Основные модули программы вместе с основными функциями показаны на рисунке 3.6, каждый из которых описан более подробно в следующих разделах.

3.3 МОДУЛЬ КАЛИБРОВКИ (CALIBRATION)

Алгоритм калибровки основан на методе Жанга [27]. В качестве калибровочного объекта используется плоская шахматная доска. Нами используется шахматная доска размером 6x8 клеток (для алгоритма калибровки размеры доски должны быть различны), распечатанная на бумаге формата А3. Для калибровки важно, чтобы калибровочный объект был твердым и не менял формы. Поэтому бумажную шахматную доску мы приклеили к стеклянной поверхности почти того же размера. Очень важно, чтобы поверхность бумаги и стекла были максимально плотно прикреплены друг к другу и не оставалось шероховатостей. В рамках работы также пробовалось делать калибровку шахматной доской, распечатанной на бумаге размером А4, но это не дало хороших результатов. Используемый нами шаблон шахматной доски показан на рисунке 3.7.

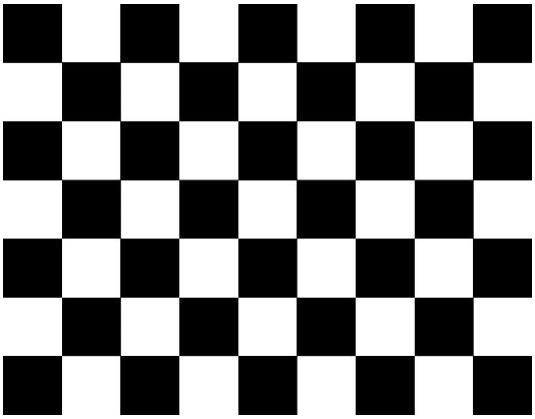


Рисунок 3.7: Шаблон шахматной доски.

Функции калибровки возвращают ошибку репроекции (re-projection error, RMS). Калибровку можно считать приемлемой (хорошей), если ошибка репроекции меньше значения 0.5.

Для достижения хороших результатов калибровки необходимо учесть несколько факторов:

- Квадраты шахматной доски должны быть четко видны и иметь четкие границы.
- Шахматная доска должна быть плотной и гладкой (ровной).
- Очень важную роль играет освещение. Комната должна быть хорошо и равномерно освещена.
- На изображениях, отснятых камерой, шахматная доска должна иметь различные ориентации и позиции.

Откалибровать стереокамеру означает вычислить следующие параметры:

- CM1 и CM2 - матрицы внутренних параметров левой и правой камеры, которые имеют вид: $\begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$

- D1 и D2 - коэффициенты искажений для обоих камер, которые имеют вид: $k_1, k_2, k_3, p_1, p_2, k_4, k_5, k_6$
- R - матица вращения
- T - вектор сдвигов
- E - существенная матрица
- F - фундаментальная матрица

Существует два подхода решения задачи стереокалибровки. Первый подход вычисляет все вышеуказанные параметры сразу из последовательности изображений калибровочного объекта, отснятого двумя камерами. Второй подход заключается в том, что сначала нужно откалибровать каждую камеру в отдельности (то есть вычислить матрицы камер и коэффициенты искажений для этих камер) и лишь потом на основе их внутренних параметров вычислить остальные параметры. Предпочтительно использовать второй метод, так как функции стереокалибровки не приходится оптимизировать внутренние параметры камер.

Для калибровки камеры необходимо около 10-15 фотографий шахматной доски. Шахматная доска должна быть видна в различных позициях и с разных ракурсов. На рисунках 3.8 и 3.9 приведены примеры плохой и хорошей конфигурации шахматной доски для калибровки соответсвенно. Больше примеров можно найти в [58].

Рисунок 3.8: Плохая конфигурация: шахматные доски маленькие (находятся слишком далеко) и почти все с похожими ориентациями [58].

Рисунок 3.9: Хорошая конфигурация: Шахматные доски в различных ориентациях и почти полностью покрывают изображение [58].

Функции калибровки вычисляют вышеуказанные параметры и записывают их в файле формата YAML [59], который имеет расширение .yml. YAML - язык для сериализации данных, удобный для ввода/вывода типичных структур данных, в частности, объектов класса OpenCV Mat. Ниже приведен пример, созданного нами .yml файла, где хранятся калибровочные данные:

```
%YAML:1.0
CM1: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 6.1538318379930513e+002, 0., 3.2475993626786959e+002, 0.,
      6.1538318379930513e+002, 2.3849970629528039e+002, 0., 0., 1. ]
CM2: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 6.1538318379930513e+002, 0., 3.2293214904864459e+002, 0.,
      6.1538318379930513e+002, 2.3997887747578062e+002, 0., 0., 1. ]
D1: !!opencv-matrix
  rows: 1
```

```
  cols: 8
  dt: d
  data: [ -1.7318315193495869e-003, 6.1062937571629150e-001, 0., 0., 0.,
      0., 0., 2.0726531212949593e+000 ]
D2: !!opencv-matrix
  rows: 1
  cols: 8
  dt: d
  data: [ 1.5377030146941642e-001, -8.3573456883075370e-001, 0., 0., 0.,
      0., 0., -2.4733278347121468e+000 ]
R: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 9.9973290472444365e-001, 1.5772186998558801e-003,
      -2.3057137558664079e-002, -1.1127730111894208e-003,
      9.9979650563878375e-001, 2.0142220550448276e-002,
      2.3084214248093898e-002, -2.0111183298109469e-002,
      9.9953121980201187e-001 ]
T: !!opencv-matrix
  rows: 3
  cols: 1
  dt: d
  data: [ -1.2470077299209322e+001, -1.2793180405600296e-001,
      1.5171886762576831e-001 ]
E: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ -2.7843765127917651e-003, -1.4911513373068824e-001,
      -1.3092778705295108e-001, 4.3954028029825287e-001,
      -2.5054871647085225e-001, 1.2460733371102979e+001,
      1.4177399954155218e-001, -1.2467337932361339e+001,
      -2.5412478844605990e-001 ]
F: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
```

data: [2.7626823251914682e-008, 1.4795331826847068e-006,
4.3758968051855758e-004, -4.3611564672054881e-006,
2.4859659152181564e-006, -7.5260376734628667e-002,
1.7200812384588634e-004, 7.5049762041012283e-002, 1.]

CM1 - это матрица первой камеры. «!!opencv-matrix» - означает, что это матрица типа Mat (класс OpenCV). В data записаны значения матрицы.

Фрагмент кода - функция, которая выполняет стерео калибровку приведен в приложении.

Рассмотрим ниже основные функции модуля.

Основные функции:

- calibrateCamera() - калибрует камеру и сохраняет внутренние параметры камеры в .yml файле.
- stereoCalibrate() - стереокалибрует две камеры, вычисляя все внутренние и внешние параметры камер, сохраняя их в файле.
- stereoCalibrateExtrinsic() - стереокалибрует две камеры, вычисляя внешние параметры камер на основе заранее вычисленных внутренних параметров камер, которые могут быть вычислены при помощи функции calibrateCamera.
- captureImages() - захватывает изображения в режиме реального времени одновременно для двух камер.

3.4 МОДУЛЬ РЕКТИФИКАЦИИ (RECTIFCATION)

Во время ректификации на основе данных CM1, CM2, D1, D2, R, T вычисляются параметры: R1, R2, P1, P2 и Q. R1 и R2 - ректификационные преобразования (матрицы поворотов) для первой и второй камеры. P1 и P2 - матрицы проекции в новой (ректифицированной) системе координат для первой и второй камеры. Q - матрица соответствия смещение-глубина (disparity-to-depth mapping matrix).

Матрицы $P1$ и $P2$ имеют следующий вид:

$$P1 = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ и } P2 = \begin{bmatrix} f & 0 & cx_2 & T_x f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

где f - фокусное расстояние, $cx_1, cx_2, cy_1 = cy_2 = cy$ - координаты центров проекций, T_x - расстояние между камерами.

Матрица Q имеет вид:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -cx_1 \\ 0 & 1 & 0 & -cy \\ 0 & 0 & -\frac{1}{T_x} & \frac{(cx_1 - cx_2)}{T_x} \end{bmatrix}$$

Матрица Q используется при триангуляции для получения карты глубины (облака точек).

На рисунке 3.10 приведен пример вычисленных эпиполярных линий для стереопары изображений, которые используются при ректификации. А на рисунке 3.11 ректифицированые изображения.

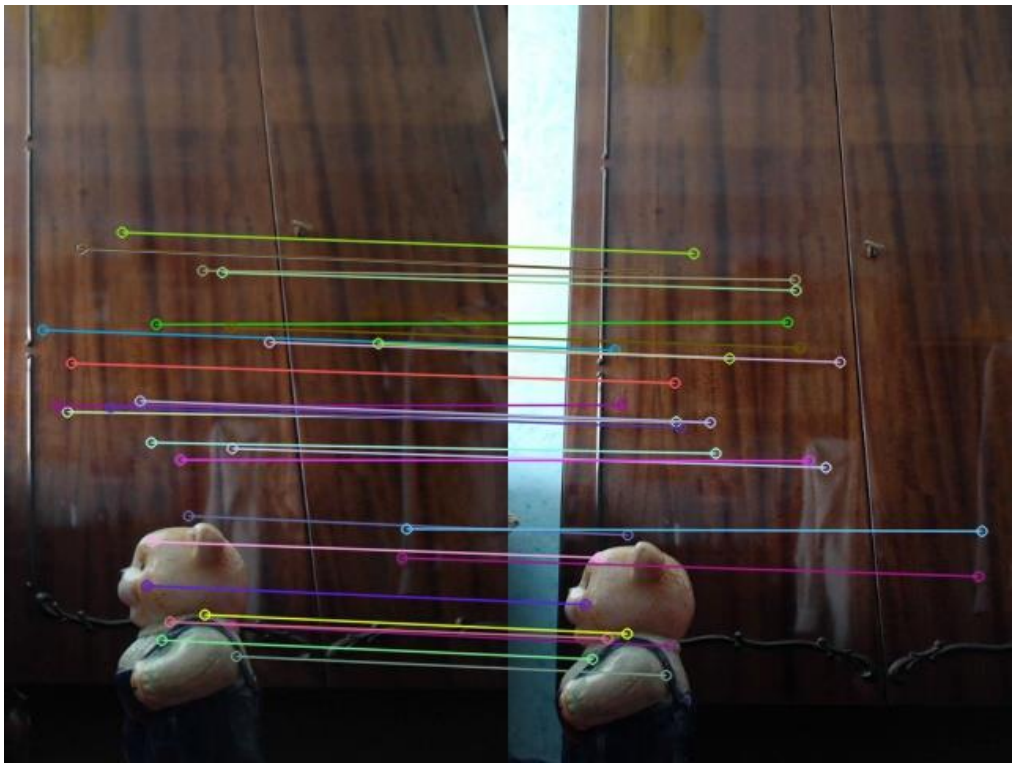


Рисунок 3.10: Пример вычислимых эпиполярных линий.

Рисунок 3.11: Пример ректифицированный стереопары.

Основные функции:

- stereoRectify() - на основе вычисленных параметров ректификации, ректифицирует стереопару (изображения, одновременно снятые левой и правой камерой).
- stereoRectifySURF() - ректифицирует стереопару, не зная параметры камер. Ректификация выполняется на основе дескриптора характеристик SURF. Данная функция, в основном дает результаты хуже, чем функция stereoRectify() и может быть использована только в том случае, когда неизвестны параметры камер.

3.5 МОДУЛЬ ПОСТРОЕНИЯ КАРТЫ СМЕЩЕНИЙ (DISPARITY MAP)

Карта смещений строится на основе алгоритма SGM (Semi-Global Matching), который был описан в первой главе (раздел 1.7). Также были протестированы алгоритмы Stereo Variation и Block Matching, но наилучший результат дал алгоритм SGM, поэтому в работе используется данный алгоритм.

Очень важную роль для алгоритма SGM играет выбор значения следующих параметров:

- minDisparity - минимальное возможное значение смещения (диспаритета). Значение обычно бывает равными нулю, но в некоторых случаях алгоритмы ректификации могут сдвинуть изображения, тогда данный параметр должен быть установлен соответственно.
- numDisparities - это разница максимального и минимального значения смещений. Данное значение всегда больше нуля. В данной имплементации этот параметр должен быть кратен 16-ти.
- SADWindowSize - размер сопоставимого блока. Значение должно быть нечетное число, которое больше или равно 1. Обычно значение параметра бывает в диапазоне от 3 до 11.

- P1 - налагаемый штраф в том случае, если у соседних пикселей смещение отличается на 1. Возможное значение параметра: *8*n*SADWindowSize* SADWindowSize*, где n– количество каналов изображения.
- P2 - штраф, если у соседних пикселей смещение отличается больше чем на 1. Алгоритм требует, чтобы P2 > P1. Параметры P1 и P2 контролируют гладкость карты смещений. Возможное значение параметра: *32*n*SADWindowSize* SADWindowSize*.
- disp12MaxDiff - максимально допустимая разница в проверке левого-правого смещения. Если значение параметра не положительное, то проверка не выполняется.
- preFilterCap - значение сокращения для пикселей предварительно отфильтрованного изображения. Алгоритм вначале считает производную по направление x для каждого пикселя и берет те значения, которые находятся в интервале [-preFilterCap, preFilterCap]. Данные значения передаются методу Birchfield-Tomasi [35], который вычисляет попиксельную стоимость.
- uniquenessRatio – число в процентах, на которое минимальное (лучшее) значение, вычисляемое функцией стоимости, должно превосходить второе минимальное значение, чтобы соответствие было правильным. В основном, для хороших результатов значение параметра должно быть в районе 5-15.
- speckleWindowSize - максимальный размер регионов гладкости, который используется фильтром удаления шумов. Если значение параметра равно 0, то фильтр не применяется, в противном случае, значение должно быть установлено в районе 50-200.
- speckleRange - максмальное значение изменения смещения внутри каждой связанной компоненты. Значение параметра алгоритмом умножается на 16. Обычно значения 1 и 2 являются хорошими.
- fullDP - если значение данного параметра равно true, то работает полномасштабный двухпроходной алгоритм динамического программирования.

В данном случае количество операций будет O(W*H*numDisparities), где W и H - размеры изображения. По умолчанию значения параметра равно false.

На рисунке 3.12 приведен скриншот части нашей программы, которая дает возможность пользователю изменить параметры алгоритма SGM и посмотреть, как это влияет на карту смещений.

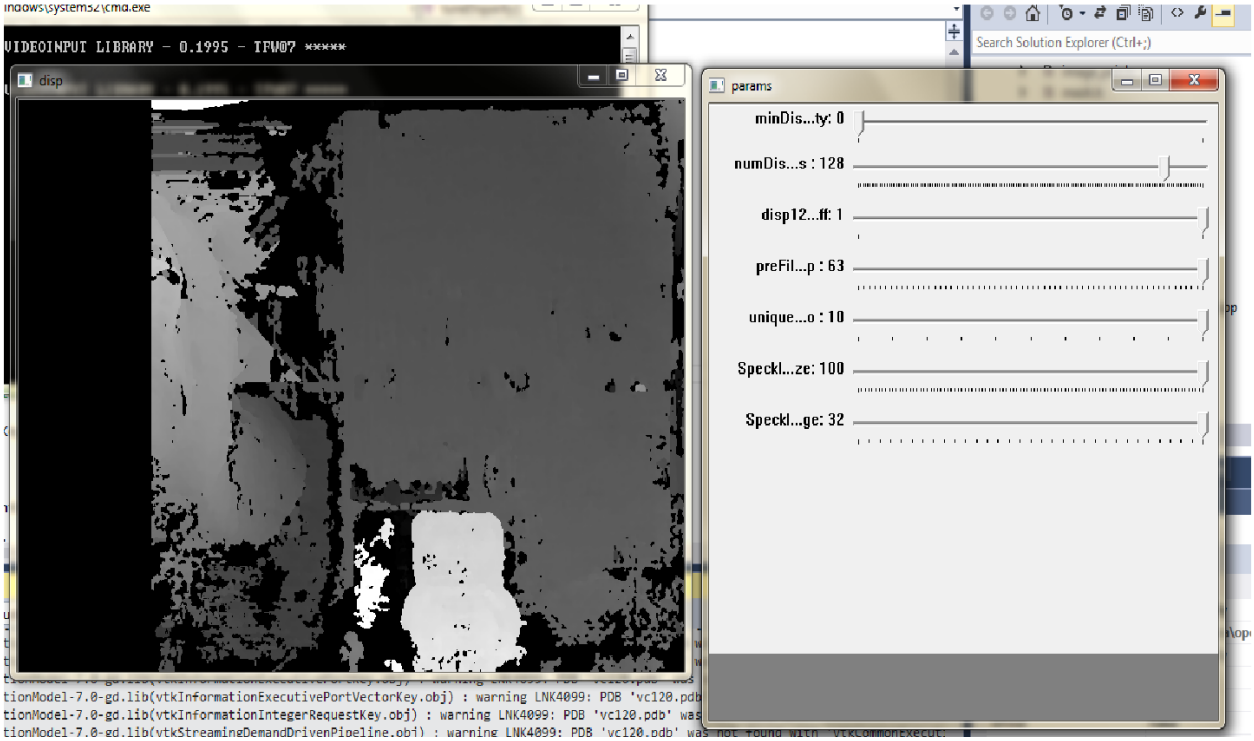


Рисунок 3.12: Настройка параметров SGM.

Основные функции:

- getDisparityMap() - из стереопары изображений вычисляет и возвращает карту смещений.
- tuneDisparityMap() - как было упомянуто, для алгоритма стерео соответствия SGM важную роль играет выбор значения параметров. Данная функция дает пользователю графическую возможность изменить значения параметров и посмотреть, как это влияет на вычисление карты смещений.

3.6 МОДУЛЬ ПОСТРОЕНИЯ КАРТЫ ГЛУБИНЫ / ОБЛАКО ТОЧЕК (DEPTH MAP)

Данный модуль по карте смещений и по параметрам камер, при помощи триангуляции вычисляет карту глубины. Карта глубины сразу преобразуется в облако точек, хотя может быть сохранена и в других форматах.

На рисунке 3.13 показано облако точек всей сцены, а на рисунке 3.14 облако точек объекта отделенное от сцены.

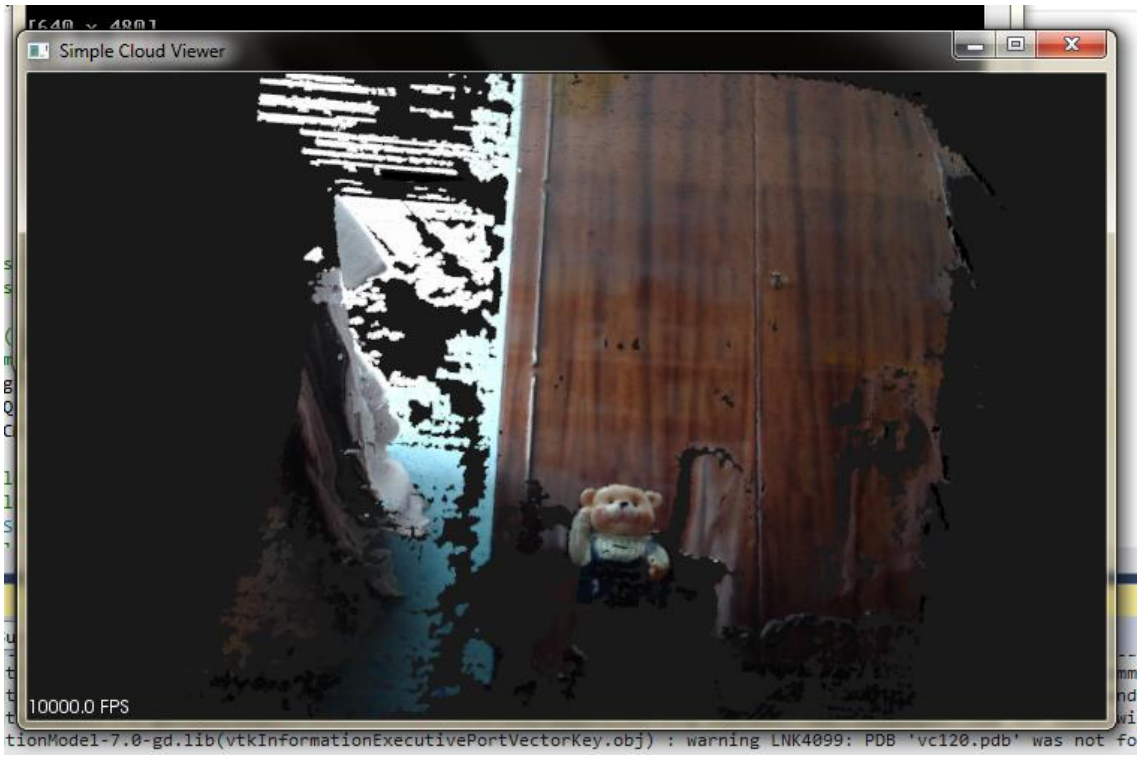


Рисунок 3.13: Облако точек всей сцены.

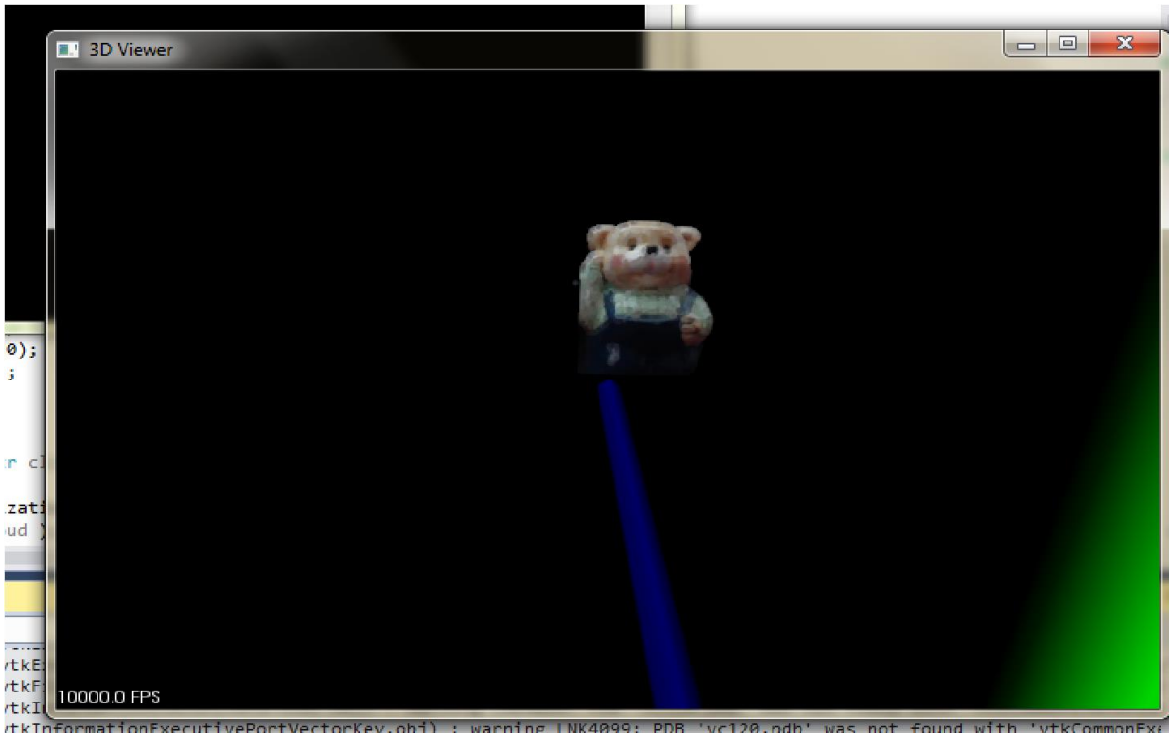


Рисунок 3.14: Облако точек объекта.

Основные функции:

- getDepthMap() - возвращает карту глубины в виде облака точек
- showCoordinates() - показывает пользователю исходное изображение левой камеры, и дает пользователю возможность выбирать пиксели изображения. Для выбранных пикселей выводятся на экран их трехмерные координаты X, Y и Z.
- savePointClouds() - для заданного множества стереопар функция вычисляет и сохраняет соответствующие им облака точек. Перед сохранением облака точек, применяется фильтр прохождения-через (pass-through filter), который отбрасывает все точки, координаты которых не находятся в заданном диапазоне. Для каждой координаты задается свой диапазон. Выбрав правильные диапазоны, можно отделить объект от сцены. Данная функция используется для получения облаков точек разных ракурсов объекта, которые потом используются при регистрации.

3.7 МОДУЛЬ ОБРАБОТКИ ОБЛАКОВ ТОЧЕК (POINT CLOUDS PROCESSING)

В данном модуле содержатся различные вспомогательные функции для работы с облаками точек, которые в основном используются для обработки облаков точек. Эти функции выполняют различные фильтры и вычисляют дополнительную информацию, такую как нормали облака точек.

Основные функции:

- getNormals() - вычисляет нормали для заданного облака точек.
- downsample() - выполняет даунсамплинг, то есть уменьшает разрешения облака точек. Данная функция уменьшает количество точек в облаке точек, что положительно влияет на эффективность.
- passThroughFilter() - данная функция для заданного поля (координаты), отбрасывает все точки, для которых значение этого поля не входит в заданный диапазон.
- statisticalOutlierFilter() - применяет фильтр удаления выбросов (удаленных точек) на основе статистики (SOR) к облаку точек.
- filterBlackPoints() - в некоторых случаях, в облаке точек присутствуют лишние черные точки. Данная функция удаляет черные точки из облака точек.
- smoothing() - делает облако точек более гладким.
- bilateralFilter() - применяет билатеральный фильтр (bilateral filter) к облаку точек.

3.8 МОДУЛЬ РЕГИСТРАЦИИ (REGISTRATION)

В данной работе реализован довольно гибкий модуль для регистрации облаков точек, который дает возможность комбинировать различные алгоритмы регистрации облаков точек, а также легко добавлять новые. Данный модуль также реализует разработанный метод регистрации, описанный в главе 2.

Данный модуль можно разделить на две части: функции, которые выполняют начальную (грубую) регистрацию и конечную (fine) регистрацию.

В качестве алгоритма начальной регистрации нами был реализован метод регистрации на основе характеристик, который описан в разделе 2.3.

Алгоритм состоит из следующих частей:

- вычисление ключевых точек
- вычисление дескрипторов характеристик
- поиск соответствий
- отбрасывание плохих соответствий
- поиск преобразования

Каждая из частей реализована в виде отдельной функции, и поэтому может быть легко заменена на другую реализацию.

Псевдокод функции, которая выполняет регистрацию на основе характеристик показан на рисунке 3.15.

```
01. function registerPairFb()
02. {
03.     points1 <- calcKeyPoints(cloud1)
04.     points2 <- calcKeyPoints(cloud2)
05.     feautures1 <- getFeautures(points1)
06.     feautures2 <- getFeautures(points2)
07.     allCorrespondences <- getCorrespondences(feautures1, feautures2)
08.     goodCorrespondecnes <- rejectBadCorrespondences(allCorrespondences)
09.     transformation <- getTransformation(cloud1, cloud2, goodCorrespondecnes)
10.     cloud1 <- transformCloud(cloud1, transformation)
11.     resultCloud <- cloud1 + cloud2
12.     return resultCloud
13. }
```

Рисунок 3.15: Псевдокод регистрации на основе характеристик.

Строки 3 и 4 вычисляют ключевые точки для первого и второго облака точек. Строки 5 и 6 вычисляют дескрипторы характеристик, в нашем случае при помощи FPFH дескриптора. Далее строка 7 на основе характеристик обоих облаков точек вычисляет соответствия. Строка 8 фильтрует соответствия. Далее на основе вычисленных соответствий вычисляется преобразование, которое приводит координаты первого облака точек к координатам второго. Строка 10 применяет вычисленное преобразование к первому облаку точек. Далее облака точек соединяются в одно облако точек, которое возвращается в качестве результата.

Вместо вычисления ключевых точек, мы используем все точки из облака точек, поскольку наши регистрируемые облака точек не содержат слишком много точек. Были протестированы различные дескрипторы характеристик и на основе экспериментальных данных был выбран дескриптор FPFH.

Начальная регистрация не является обязательным шагом. В случае, когда облака точек находятся близко по отношению друг к другу (смещены на маленькое расстояние и повернуты на маленький угол) иногда бывает лучше сразу применить алгоритм ICP.

Нами реализована предложенная модификация алгоритма ICP, которая описана в разделе 2.5. Псевдокод функции, которая реализует регистрацию двух облаков точек на основе алгоритма ICP, показан на рисунке 3.16.

```
01. function registerPairICP()
02. {
03.     while(!converged())
04.     {
05.         allCorrespondences <- findCorrespondences(cloud1, cloud2)
06.         goodCorrespondecnes <- rejectBadCorrespondences(allCorrespondences)
07.         transformation <- findTransformation(cloud1, cloud2, goodCorrespondecnes)
08.         cloud1 <- transformCloud(cloud1, transformation)
09.     }
10. }
```

Рисунок 3.16: Псевдокод регистрации при помощи ICP.

Строка 3 проверяет условие прерывания выполнения алгоритма ICP. Данным условием может быть:

- Изменение ошибки для двух последовательных итераций меньше некоторого заданного значения.
- Превышение максимального количества итераций.

Строка 5 находит соответствия между облаками точек по принципу ближайших точек. Для быстрого поиска соответствий используются k-d деревья. Строка 6 отбрасывает плохие соответствия. Строка 7 находит преобразование, применяя которое к первому облаку точек уменьшает расстояние между облаками точек. Строка 8 применяет данное преобразование к первому облаку точек. Для вычисления преобразования, были протестированы различные метрики и на основе экспериментальных данных была выбрана метрика точка-плоскость с оптимизацией наименьших квадратов, которая описана в разделе 2.5.

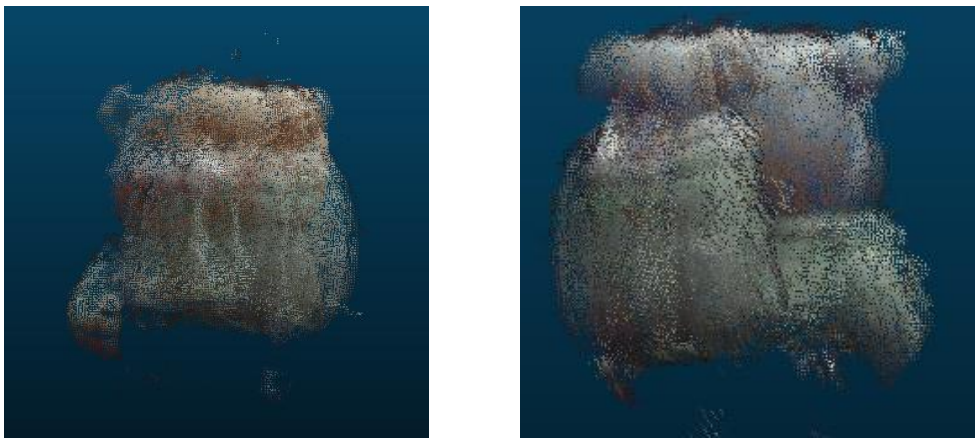
На таблице 3.1 ниже приведено сравнение нескольких версий (модификаций) алгоритма ICP. Верхняя строка показывает методы вычисления соответствий, к которым относятся: сингулярное разложение матрицы (SVD), метод двойных кватернионов и линейная оптимизация наименьших квадратов (LLS). Первый столбец показывает применение алгоритма отбрасывание соответствий. Значения таблицы - это значение ошибки после применения алгоритма ICP, которое вычисляется как среднее расстояние между соответствующими точками.

ICP	SVD	Двойные кватерионы	LLS
Отбрасывание соответствий	0.0113504	0.0112048	0.0109859
Без отбрасывания	0.0359336	0.0359386	0.0402596

Таблица 3.1: Сравнение модификаций алгоритма ICP.

Как видно из таблицы, применение алгоритма отбрасывания плохих соответствий улучшает результаты регистрации. Как было описано в разделе 2.5 в работе используется модификация ICP, которая для вычисления преобразования использует линейную оптимизацию наименьших квадратов (LLS) с отбрасыванием соответствий (правый столбец, верхняя строка) .

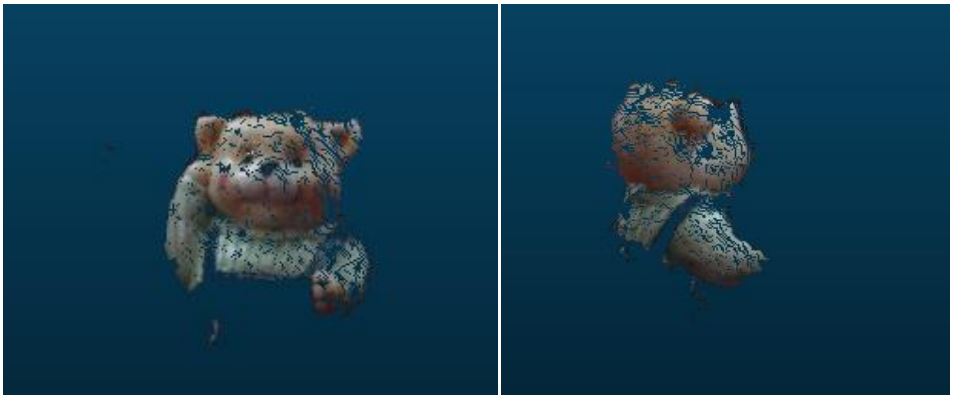
В рамках диссертационной работы было проведено множество экспериментов с различными комбинациями и модификациями алгоритмов регистрации, но разработанный алгоритм регистрации дает наилучшие результаты. На рисунке 3.17 приведены примеры неудачных регистраций облаков точек.



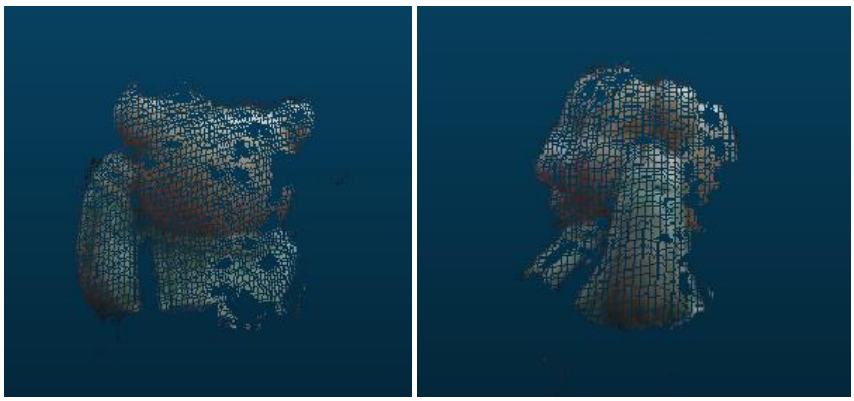
а) б)

Рисунок 3.17: Примеры недуачных регистраций.

Ниже приведен пример совмещения последовательности облаков точек различных ракурсов реального объекта в одну единую модель при помощи описанного алгоритма регистрации. На рисунке 3.18 показаны облака точек различных ракурсов объекта, а на рисунке 3.19 модель, полученная при помощи соединения этих облаков точек.

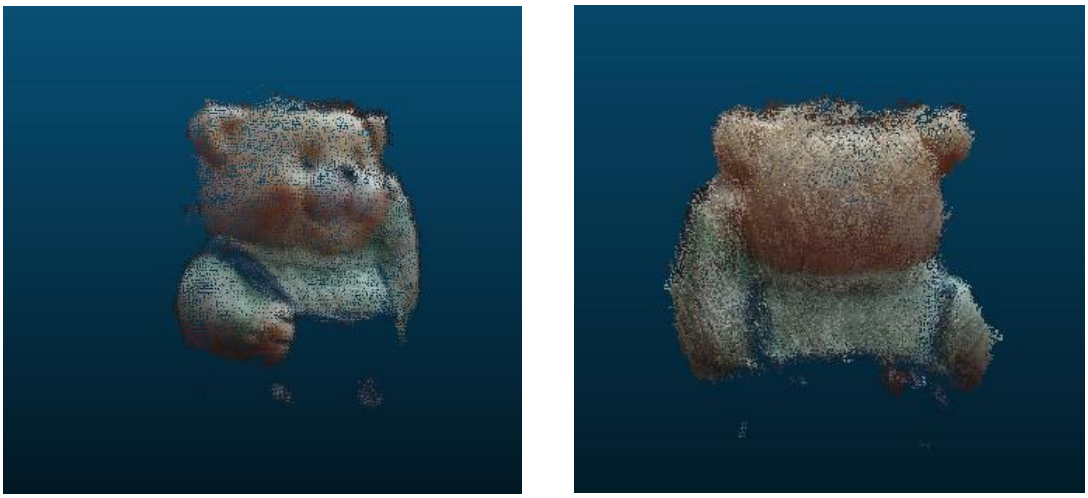


а) б)

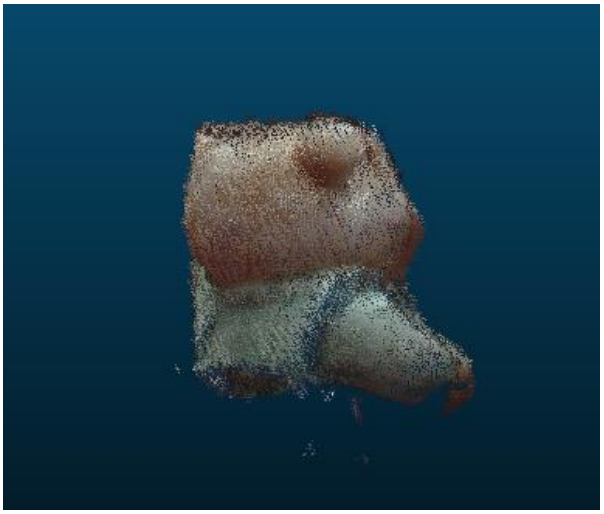


в) г)

Рисунок 3.18: Облака точек объекта медведя с различных ракурсов.



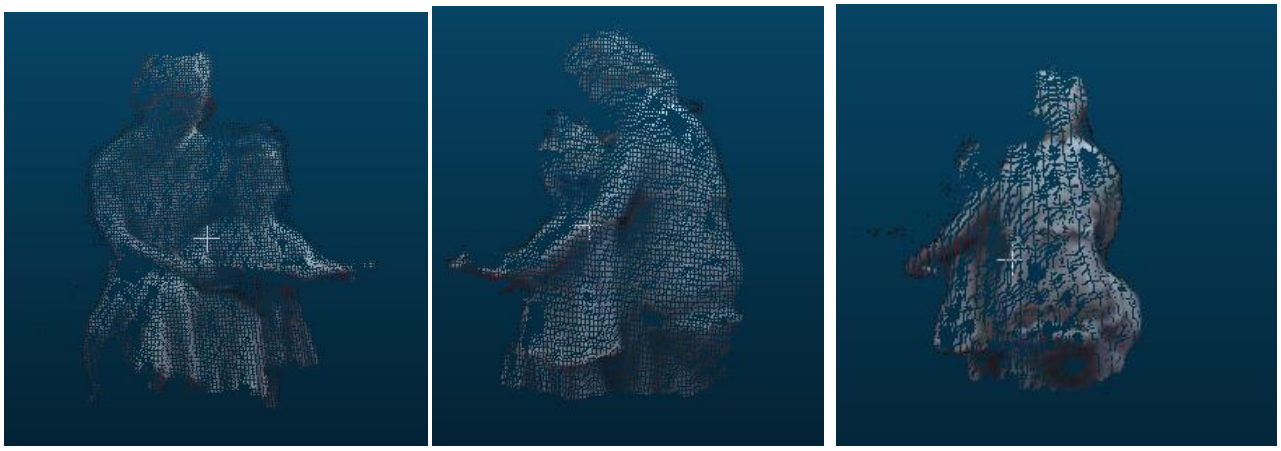
а) б)



в)

Рисунок 3.19: 3D модель медведя после совмещения (регистрации) облаков точек.

На рисунке 3.21 приведен пример регистрации нескольких облаков точек объекта статуи, которые приведены на рисунке 3.20. На рисунке 3.22 облака точек утки, а на рисунке 3.23 полученная 3D модель утки.



а) б) в)

Рисунок 3.20: Облака точек объекта статуи с различных ракурсов.

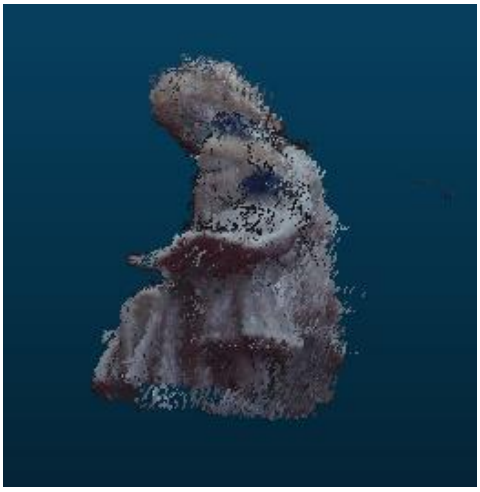


Рисунок 3.21: 3D модель статуи после регистрации облаков точек.

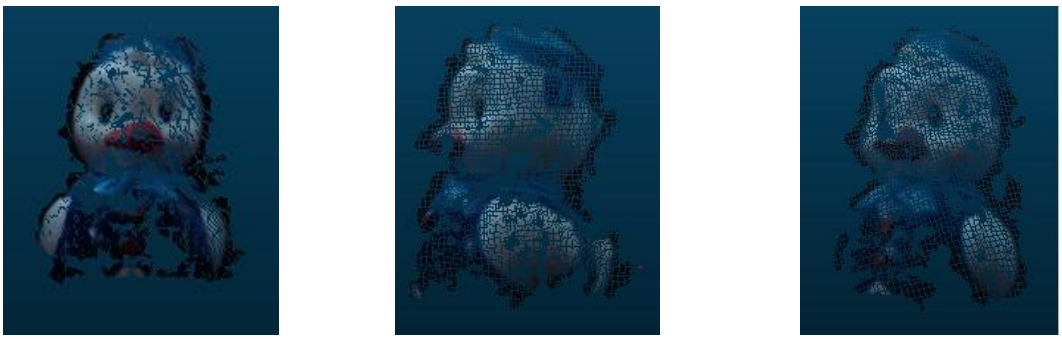


Рисунок 3.22: Облака точек объекта утки с различных ракурсов.

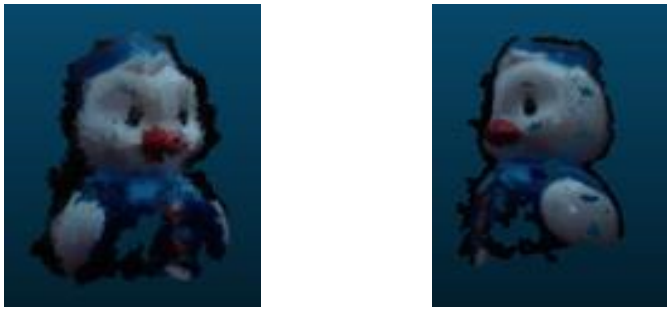


Рисунок 3.23: 3D модель утки после регистрации облаков точек.

Основные функции:

- registerPairFPFH() - соединяет пару облаков точек, при помощи алгоритма регистрации на основе характеристик.
- registerPairICP() - соединяет пару облаков точек, при помощи алгоритма ICP.
- registerPointClouds() - соединяет множество облаков точек в одно единое облако точек. Первое облако точек берется в качестве глобальной модели. По очереди берется следующее последовательное облако точек и соединяется с глобальной моделью при помощи алгоритма попарной регистрации. В зависимости от совмещаемых данных, в качестве попарной регистрации может быть сразу применена функция registerICP() или же в начале быть применена registerFPFH() (или другая функция грубой регистрации) потом registerICP().

3.9 МОДУЛЬ ВИЗУАЛИЗАЦИИ (VIZUALIZATION)

В данном модуле реализованы различные функции визуализации данных.

Основные функции:

- showPointCloud() - показывает облако точек
- showCloudNormals() - показывает нормали облака точек
- showCorrespondences() - показывает соответствия между двумя облаками точек

3.10 МОДУЛЬ ВОССТАНОВЛЕНИЯ ПОВЕРХНОСТИ (MESH PROCESSING)

В данном модуле реализованы функции, которые восстанавливают поверхность из облака точек и строят полигональные сети (polygonal mesh). Построенная полигональная сеть может быть записана в .stl файле. Формат файла STL является одним из наиболее популярных форматов для хранения трехмерных моделей. STL файл может быть бинарным (binary) или текстовым (ASCII). Пример полигональной модели, построенной по облаку точек показан на рисунке 3.22.

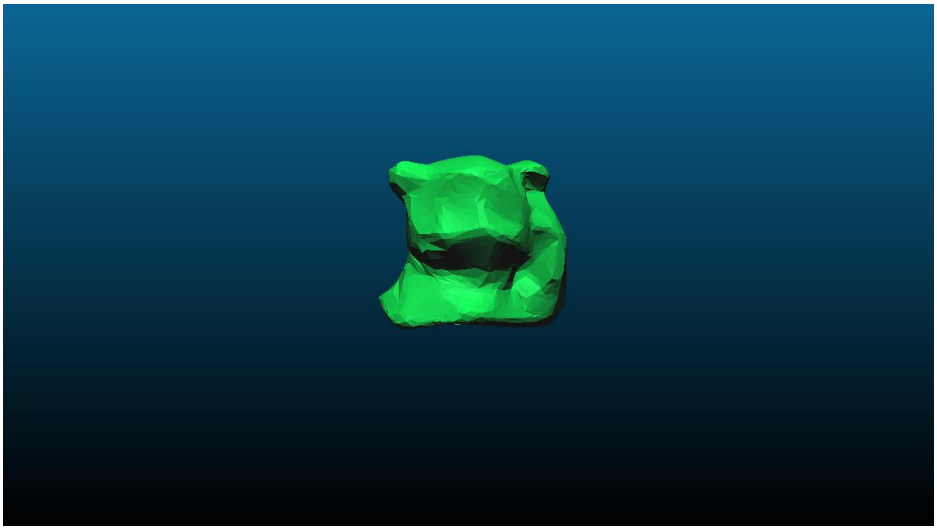


Рисунок 3.22: Пример полигональной модели, построенной по облаку точек.

Основные функции:

- greedyTriangulationReconstruction() - возвращает полигональную модель, построенную из облака точек при помощи алгоритма Greedy Triangulation.
- poissonReconstruction() - возвращает полигональную модель, построенную из облака точек при помощи алгоритма Poisson Reconcstruction.
- marchingCubesReconstruction() - возвращает полигональную модель, построенную из облака точек при помощи алгоритма Marching Cubes.

3.11 3D ПЕЧАТЬ

Одно из возможных применений полученной 3D модели - это использовать ее для 3D печати. Полученная нами трехмерная модель была распечатана при помощи 3D принтера. Тем самым было показано, что реализованное программное обеспечение дает возможность построить 3D модель реального физического объекта и распечатать ее.

Процесс 3D печати происходит следующим образом.

Трехмерная модель, записанная в STL файле передается программе Repetier-Host [60], которая преобразует файл в G-Code. G-Code - язык управления (программирования) устройств с числовым программным управлением (ЧПУ). G-Code используется большинством 3D принтеров для печати.

Пример преобразования 3D модели в G-Code при помощи программы Repetier-Host приведен на рисунке 3.23.

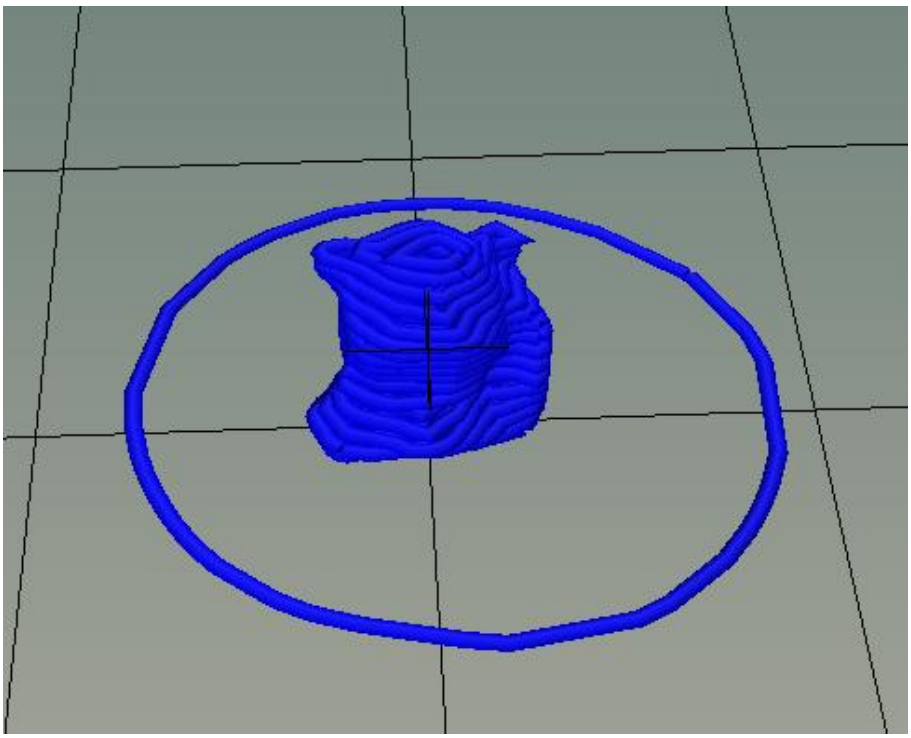


Рисунок 3.23: Преобразование 3D модели в G-Code в виде слоев программой Repetier-Host.

При помощи G-Code трехмерная модель разделяется на слои. Далее 3D принтер послойно начинает печатать 3D модель. Пример распечатанной нами модели приведен на рисунке 3.24.

Рисунок 3.24: Пример распечатанной 3D модели.

3.12 ЗАКЛЮЧЕНИЕ

В данной главе описано разработанное программное обеспечение, которое позволяет получить трехмерную модель физического объекта на основе двух камер или уже отснятой последовательности стереоизображений. Программное обеспечение реализовано на языке C++ с использованием библиотек OpenCV и PCL. Разработанная программа состоит из следующих основных модулей:

- Модуль калибровки - в режиме реального времени (видеопоследовательность с камер) или при помощи последовательности отснятых изображений калибровочного объекта (шахматной доски) вычисляет внутренние и внешние параметры камер.
- Модуль ректификации - на основе вычисленных параметров камер ректифицирует стереопару изображений.
- Модуль стерео сопоставления - по ректифицированной стереопаре изображений вычисляет карту смещений.
- Модуль построения карты глубины - на основе карты смещений и параметров камер вычисляет карту глубины (облако точек).
- Модуль обработки облака точек - содержит различные фильтры и функции для вычисления дополнительных данных, таких как нормали.
- Модуль регистрации - совмещает облака точек разных ракурсов в одно единое облако. В данном модуле реализованы функции регистрации на основе характеристик и алгоритм ICP.
- Модуль восстановления поверхности - строит полигональную сеть из облака точек.

Разработанный программный комплекс был протестирован на различных примерах, в основном с использованием двух камер Logitech C920. В частности, была получена трехмерная модель игрушечной кассы медведя и распечатана на 3D принтере. Тем самым было показано практичное применение программного комплекса.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ ДИССЕРТАЦИОННОЙ РАБОТЫ

1. Разработан алгоритм регистрации облаков точек, который исползует алгоритмы предварительной обработки облаков точек, алгоритм регистрации на основе FPFH (Fast Point Feature Histograms) дескриптора и ICP (Iterative Closest Point) алгоритм.
2. Разработан метод создания 3D модели объекта на основе стереозрения и разработанного алгоритма регистрации облаков точек.
3. Разработан программный комплекс, который позволяет получить 3D модель реального объекта. Он состоит из модулей, которые могут быть исползованы для калибровки камеры, получения карты глубины, регистрации облаков точек и т. д.

СПИСОК ЛИТЕРАТУРЫ

1. M. Shahrin, F. Hashim1, W. Zaki1, A. Hussain1, T. Raj, "3D Indoor Mapping System Using 2D LiDAR Sensor for Drones", International Journal of Engineering & Technology, pp. 179-183, 2018.
2. V. Chougule, H. Gosavi, M. Dharwadkar, A. Gaind, "Review of Different 3D Scanners and Scanning Techniques", IOSR Journal of Engineering (IOSRJEN), pp. 41-44, 2018.
3. M. Pashaei, S. Mousavi, "Implementation of a low cost structured light scanner", ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-5/W2, pp.477-482, 2013.
4. T. Lin, "Resolution adjustable 3D scanner based on using stereo cameras", Signal and Information Processing Association Annual Summit and Conference (APSIPA), pp. 1-5, 2013.
5. A. Riordan, D. Toal, T. Newe, G. Dooly, "Stereo Vision Sensing: Review of existing systems.", Twelfth International Conference on Sensing Technology (ICST), pp. 178–184, 2018
6. C. Holzmann, M. Hochgatterer, "Measuring Distance with Mobile Phones Using Single-Camera Stereo Vision", Distributed *Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on,* pp. 88-93, 18-21 June 2012.
7. D. Forsyth, J. Ponce, "Computer Vision: A Modern Approach", 2nd ed., Pearson, November 5, 2011.
8. Stereo Realist Camera: https://camerapedia.fandom.com/wiki/Stereo_Realist
9. Bumblebee: https://www.ptgrey.com/support/downloads/10132/
10. 3D Scanning with Microsoft Kinect: http://www.open-electronics.org/3d-scanning-with-microsoft-kinect/
11. Surveyor Stereo Vision System ("SVS"): http://www.robotshop.com/blog/en/add-stereo-vision-to-your-robot-with-surveyor-stereo-vision-system-3820
12. MEGA-DCS: http://users.rcn.com/mclaughl.dnai/sthmdcs.htm
13. PCI nDepth vision system: http://www.focusrobotics.com/products/systems.html

14. Minoru 3D Webcam: http://robocv.blogspot.in/2012/01/minoru-3d-webcam-for-real-time-stereo.html
15. Q. Wang, Z. Yu, C. Rasmussen, J. Yu, "Stereo vision–based depth of field rendering on a mobile device", Journal of Electronic Imaging, vol. 23, 19 March 2014.
16. Karlsruhe Dataset: Stereo Video Sequences + rough GPS Pose: http://www.cvlibs.net/datasets/karlsruhe_sequences/
17. ASL Datasets: http://projects.asl.ethz.ch/datasets/doku.php?id= laserregistration:apartment:home#downloads
18. Carnegie Mellon Image Database: http://vasc.ri.cmu.edu/idb/html/stereo/index.html
19. Datasets for disparity and optic flow evaluation: http://www.pspc.unige.it/~manuela/?page_id=102
20. MPI Sintel Stereo Training Data: http://sintel.is.tue.mpg.de/stereo
21. Сайт для сравнения алгоритмов стерео сопоставления: http://vision.middlebury.edu/stereo/
22. Tsukuba stereo pair: http://cvlab-home.blogspot.am/2012/05/h2fecha-2581457116665894170-displaynone.html
23. P. Hillman, "White paper: Camera Calibration and Stereo Vision", Lochrin Terrace, Edinburgh EH3 9QL, Tech. Rep, 2005.
24. S. Wang, Y. Geng, R. Jin, "A Novel Error Model of Optical Systems and an On-Orbit Calibration Method for Star Sensors.", Sensors, vol. 15 issue 12, pp. 31428-31441, 2015.
25. M. Axholt, "Pinhole Camera Calibration in the Presence of Human Noise", Department of Science and Technology, Linköping UniversitySE-601 74 Norrköping, Sweden 2001.
26. R. Tsai, "An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, pp. 364–374, 1986.
27. Z. Zhang, "A flexible new technique for camera calibration", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1330–1334, 2000.
28. R. Hartley, A. Zisserman, "Multiple View Geometry in Computer Vision", 2nd ed., Cambridge University Press, April 19 2004.
29. Camera Calibration Toolbox for Matlab:

http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

30. Image Rectification: http://www.sci.utah.edu/~gerig/CS6320-S2012/Materials/CS6320-CV-F2012-Rectification.pdf
31. R. Hartley, "Theory and Practice of Projective Rectification", International Journal of Computer Vision, vol. 35, pp 115–127, November 1999.
32. A. Olofsson, "Modern stereo correspondence algorithms: Investigation and evaluation", Master's thesis, Linkoping University, 2010.
33. D. Scharstei, R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms", International Journal of Computer Vision, vol. 47, pp. 7-42, 2002.
34. J. Cox, S. Hingorani, S. Rao, B. Maggs, "A maximum likelihood stereo algorithm.", CVIU, pp. 542-567, 1996
35. H. Hirschmuller, "Accurate and Efficient Stereo Processing by Semi Global Matching and Mutual Information.", IEEE Computer Vision and Pattern Recognition, vol. 2, pp. 807-814 2005.
36. J. Kim, V. Kolmogorov, R. Zabih, "Visual correspondence using energy minimization and mutual information", International Conference on Computer Vision, Vol. 2, October 2003.
37. S. Birchfield, C. Tomasi, "A Pixel Dissimilarity Measure That Is Insensitive to Image Sampling", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20 issue 4, pp. 401-406, 1998.
38. P. J.Bagga, "Real time depth computation using stereo imaging", *Journal Electrical and Electronic Engineering*, vol. 1, pp. 51-54, 2013.
39. M. A. Mahammed, A. I. Melhum and F. A. Kochery, "Object distance measurement by stereo vision", International Journal of Science and Applied Information Technology, vol. 2, pp. 05-08, March 2013.
40. База данных облаков точек Стэнфордского Университета: https://graphics.stanford.edu/data/3Dscanrep/

41. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, "Surface reconstruction from unorganized points", ACM SIGGRAPH Computer Graphics, vol. 26 issue 2, pp. 71-78, July 1992.
42. A. Alqudah, "Survey of Surface Reconstruction Algorithms", Journal of Signal and Information Processing, pp 63-79, 2014.
43. R. Rusu, N. Blodow, M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D Registration", ICRA'09 Proceedings of the 2009 IEEE international conference on Robotics and Automation, pp. 3212-3217, 2009.
44. S. Woodworth, "Reconstructing Point Clouds of Mid-size Objects", California Polytechnic State University - San Luis Obispo, December 2013.
45. The Point Cloud Library website. [Online]. Available: http://pointclouds.org/
46. X. Hana, J. Jin, J. Xie, M. Wang, W. Jiang, "A comprehensive review of 3D point cloud descriptors", CoRR, vol. abs/1802.02297, 2018.
[Online]. Available: http://arxiv.org/abs/1802.02297
47. R. Rusu, N. Blodow, Z. Marton, M. Beetz, "Aligning point cloud views using persistent feature histograms", Proc. of the 2008 International Conference on Intelligent Robots and Systems (IROS 2008), pp. 3384-3391, 2008.
48. R. Raguram, J. Frahm, M. Pollefeys, " A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus", European Conference on Computer Vision, pp. 500-513, 2008.
49. P. Besl, H. Mckay, "A method for registration of 3-Dshapes", IEEE Trans. Pattern Anal. Mach. Intell., pp. 239–256, 1992.
50. Y. Chen, G. Medioni, "Object modelling by registration of multiple range images", Image And Vision Computing, vol. 3 issue 10, pp 145–155, April 1992.
51. Zh. Zhang, "Iterative point matching for registration of free-form curves and surfaces." International Journal of Computer Vision, vol. 13, pp 119–152, October 1994.
52. A. Lorusso, D. Effert, R. Fisher, "A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations", BMVC, vol.1, pp 237-246, BMVA Press, 1995.

53. S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm", 3-D Digital Imaging and Modeling, pp. 145–152, 2001.
54. K. Low, "Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration", Techrep - Chapel Hill, University of North Carolina, 2004.
55. G. Bradski, A. Kaehler, "*Learning OpenCV: Computer Vision with the OpenCV Library*", 1st ed., O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, 2008.
56. The OpenCV website. [Online]. Available: http://opencv.org/
57. Logitech HD Pro Webcam: http://www.logitech.com/ru-ru/product/hd-pro-webcam-c920
58. Calibraiton examples: http://www.cvlibs.net/software/calibration/mistakes.php
59. YAML: http://yaml.org/
60. Repetier-Host: https://www.repetier.com/
61. А. Геворкян, «ПОЛУЧЕНИЕ 3D МОДЕЛИ ОБЪЕКТА НА ОСНОВЕ СТЕРЕОЗРЕНИЯ», in Proceedings Of National Polytechnic University of Armenia, Information Technologies, Electronics, Radio Engineering, № 2, pp. 42-49, Yerevan, Armenia, 2016.
62. Aram Gevorgyan Vladimir, "Point clouds registration and generation from stereo images", ITHEA Journal, Information Content and Processing, Vol. 3, Number 2, pp. 193-200, Bulgaria, 2016.
63. Aram V. Gevorgyan and Hakob G. Sarukhanyan, "3D Scanner from Two Web Cameras", Mathematical Problems of Computer Science 44, pp. 42-50, Yerevan, Armenia, 2015.
64. Aram Gevorgyan Vladimir, "AUTOMATIC REGISTRATION OF POINT CLOUDS WITH FPFH DESCRIPTOR", Научные горизонты,Vol. 3, pp. 55-60, Belgorod, Russia, 2017.
65. Aram V. Gevorgyan, "Point Clouds Preprocessing for Better Registration", Mathematical Problems of Computer Science 48, pp. 82-88, Yerevan, Armenia, 2017.
66. Vahan Gevorgyan Vladimir, Aram Gevorgyan Vladimir, Gevorg Karapetyan Arakel. Exemplar based inpainting using depth map information, ITHEA Journal, "Information Theories and Applications", Vol. 23, Number 3, pp. 273-281, Bulgaria, 2016.

ПРИЛОЖЕНИЕ

В данном приложении приведены некоторые фрагменты кода, а также список используемых сокращений.

Функция стерео калибровки:

```
void stereoCalibrate()
 {
	int numBoards = 15;
	int board_w = 8;
	int board_h = 6;
	Size board_sz = Size(board_w, board_h);
	int board_n = board_w*board_h;
	vector<vector<Point3f> > object_points;
	vector<vector<Point2f> > imagePoints1, imagePoints2;
	vector<Point2f> corners1, corners2;
	vector<Point3f> obj;

	double squareSize = 4.1;
	for(int i=0; i < board_h; ++i)
	{
		for(int j=0; j<board_w; ++j)
		{
			obj.push_back(Point3f(i*squareSize,j*squareSize, 0.0f));
		}
	}
	Mat img1, img2, gray1, gray2;
	int driver = CV_CAP_DSHOW;
	VideoCapture cap1(0+driver);
	VideoCapture cap2(1+driver);
	cap1.set(CV_CAP_PROP_AUTOFOCUS, false);
	cap2.set(CV_CAP_PROP_AUTOFOCUS, false);
	cap1.set(CV_CAP_PROP_SETTINGS, 1);
	cap2.set(CV_CAP_PROP_SETTINGS, 1);

	int success = 0, k = 0;
	bool found1 = false, found2 = false;
	namedWindow("left 1");
	namedWindow("right 1");
	while (success < numBoards)
	{
		cap1.grab();
		cap2.grab();
		cap1.retrieve(img1);
		cap2.retrieve(img2);
		waitKey(500);
		if(img1.empty() || img2.empty())
			continue;
		cvtColor(img1, gray1, CV_BGR2GRAY);
		cvtColor(img2, gray2, CV_BGR2GRAY);
		found1 = findChessboardCorners(img1, board_sz, corners1,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS | CV_CALIB_CB_NORMALIZE_IMAGE );
		found2 = findChessboardCorners(img2, board_sz, corners2,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS | CV_CALIB_CB_NORMALIZE_IMAGE );
		if (found1)
		{
```

```
                    cornerSubPix(gray1, corners1, Size(11, 11), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
                }
                if (found2)
                {
                    cornerSubPix(gray2, corners2, Size(11, 11), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
                }
                if(!img1.empty())
                    imshow("left 1", img1);
                if(!img2.empty())
                    imshow("right 1", img2);

                k = cv::waitKey(2);

                if (found1  && found2)
                {
                    cout<<"founded: "<< success <<endl;
                    imwrite("calib/left" + to_string((long long)success) + ".jpg", img1);
                    imwrite("calib/right" + to_string((long long)success) + ".jpg", img2);
                    imagePoints1.push_back(corners1);
                    imagePoints2.push_back(corners2);
                    object_points.push_back(obj);
                    success++;
                    if (success >= numBoards)
                    {
                        break;
                    }
                }
            }
            cap1.release();
            cap2.release();
            destroyAllWindows();
            printf("Starting Calibration\n");
            Mat CM1 = Mat(3, 3, CV_64FC1);
            Mat CM2 = Mat(3, 3, CV_64FC1);
            Mat D1, D2;
            Mat R, T, E, F;

            double calibErr = stereoCalibrate(object_points, imagePoints1, imagePoints2,
            CM1, D1, CM2, D2, img1.size(), R, T, E, F,

             CALIB_FIX_ASPECT_RATIO +
                                    CALIB_ZERO_TANGENT_DIST +
                                    CALIB_SAME_FOCAL_LENGTH +
                                    CALIB_RATIONAL_MODEL +
                                                        CALIB_FIX_K3 + CALIB_FIX_K4 +
CALIB_FIX_K5, TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 100, 1e-5));
            FileStorage fs1("calibration.yml", FileStorage::WRITE);
            fs1 << "CM1" << CM1;
            fs1 << "CM2" << CM2;
            fs1 << "D1" << D1;
            fs1 << "D2" << D2;
            fs1 << "R" << R;
            fs1 << "T" << T;
            fs1 << "E" << E;
            fs1 << "F" << F;
            fs1 << "Error" << calibErr;
            printf("Calibration error: %f \n", calibErr);
            printf("Done Calibration\n");
            printf("Starting Rectification\n");
            Mat R1, R2, P1, P2, Q;
            stereoRectify(CM1, D1, CM2, D2, img1.size(), R, T, R1, R2, P1, P2, Q);
            fs1 << "R1" << R1;
```

```
        fs1 << "R2" << R2;
        fs1 << "P1" << P1;
        fs1 << "P2" << P2;
        fs1 << "Q" << Q;
        printf("Done Rectification\n");

        return;
 }
```

Функция по стереопаре изображений вычисляет и возвращает облако точек:

```
pcl::PointCloud<pcl::PointXYZRGB>::Ptr getPointCloud()
{
        cv::Mat img1, img2;
        img1 = cv::imread("images/left1.png");
        img2 = cv::imread("images/right1.png");

        cv::Mat Q;
        cv::Mat CM1, CM2, D1, D2, P1, P2, R1, R2;

        cv::FileStorage fs("calibration.yml", cv::FileStorage::READ);
        fs["Q"] >> Q;
        fs["CM1"] >> CM1;
        fs["CM2"] >> CM2;
        fs["D1"] >> D1;
        fs["D2"] >> D2;
        fs["P1"] >> P1;
        fs["P2"] >> P2;
        fs["R1"] >> R1;
        fs["R2"] >> R2;
        fs.release();

        cv::Mat map1x, map1y, map2x, map2y;
        cv::Mat img1U, img2U;
        cv::initUndistortRectifyMap(CM1, D1, R1, P1, img1.size(), CV_32FC1, map1x, map1y);
        cv::initUndistortRectifyMap(CM2, D2, R2, P2, img2.size(), CV_32FC1, map2x, map2y);

        remap(img1, img1U, map1x, map1y, cv::INTER_LINEAR, cv::BORDER_CONSTANT,
cv::Scalar());
        remap(img2, img2U, map2x, map2y, cv::INTER_LINEAR, cv::BORDER_CONSTANT,
cv::Scalar());

        cv::Mat g1, g2, disparity;
        cvtColor(img1U, g1, CV_BGR2GRAY);
        cvtColor(img2U, g2, CV_BGR2GRAY);

        disparity = calcStereoSGBM(g1, g2);

        cv::imshow("disp", disparity);
        cv::waitKey(0);

        pcl::PointCloud<pcl::PointXYZRGB>::Ptr points = reproject3D(img1U, disparity, Q);

        return points;
}
```

функция по карте смещений вычисляет облако точек:

```
pcl::PointCloud<pcl::PointXYZRGB>::Ptr reproject3D(cv::Mat img, cv::Mat disp, cv::Mat Q)
{
	int i,j;
	vector<vector<cv::Point3d>> points;
	points.resize(disp.rows);
	double Q03, Q13, Q23, Q32, Q33;
	Q03 = Q.at<double>(0,3);
	Q13 = Q.at<double>(1,3);
	Q23 = Q.at<double>(2,3);
	Q32 = Q.at<double>(3,2);
	Q33 = Q.at<double>(3,3);
	double px,py,pz;

	pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZRGB>);
	uchar pr,pb,pg;
	for(i=0;i<disp.rows;++i)
	{
		uchar* disp_ptr = disp.ptr<uchar>(i);
		uchar* img_ptr = img.ptr<uchar>(i);
		for(j=0;j<disp.cols;++j)
		{
			uchar d = disp_ptr[j] ;
			if(d==0)
			{
				continue;
				pcl::PointXYZRGB point;
				point.x = 0;
				point.y = 0;
				point.z = 0;
				point.rgb = 0;
				cloud->points.push_back(point);
				continue;
			}

			double pw = static_cast<double>(d) * Q32 + Q33;
			px = static_cast<double>(j) + Q03;
			py = static_cast<double>(i) + Q13;
			pz = Q23;

			px = -px/pw;
			py = -py/pw;
			pz = pz/pw;

			pcl::PointXYZRGB point;
			point.x = px;
			point.y = py;
			point.z = pz;

			pb = img_ptr[3*j];
			pg = img_ptr[3*j+1];
			pr = img_ptr[3*j+2];
			uint32_t rgb = (static_cast<uint32_t>(pr) << 16 |
					static_cast<uint32_t>(pg) << 8 |
static_cast<uint32_t>(pb));
			point.rgb = *reinterpret_cast<float*>(&rgb);

			cloud->points.push_back (point);
		}
	}

	cloud->width = (int) cloud->points.size();
```

```
	cloud->height = 1;
	return cloud;
}
```

пример кода - функция, которая выполняет ICP регистрацию

```
void registerPairICP(const PointCloud<PointT>::Ptr &src, const PointCloud<PointT>::Ptr &tgt,
Eigen::Matrix4d &transform)
{
	CorrespondencesPtr all_correspondences(new Correspondences),
		good_correspondences(new Correspondences);

	PointCloud<PointT>::Ptr output(new PointCloud<PointT>);
	*output = *src;

	Eigen::Matrix4d final_transform(Eigen::Matrix4d::Identity());

	int iterations = 0;
	DefaultConvergenceCriteria<double> converged(iterations, transform,
*good_correspondences);


	// ICP loop
	do
	{
		// Find correspondences
		findCorrespondences(output, tgt, *all_correspondences);
		PCL_DEBUG("Number of correspondences found: %d\n", all_correspondences-
>size());

		if (rejection)
		{
			// Reject correspondences
			rejectBadCorrespondences(all_correspondences, output, tgt,
*good_correspondences);
			PCL_DEBUG("Number of correspondences remaining after rejection: %d\n",
good_correspondences->size());
		}
		else
			*good_correspondences = *all_correspondences;

		// Find transformation
		findTransformation(output, tgt, good_correspondences, transform);

		// Obtain the final transformation
		final_transform = transform * final_transform;

		// Transform the data
		transformPointCloudWithNormals(*src, *output, final_transform.cast<float>());

		cerr.precision(15);
		//std::cerr << transform << std::endl;

		// Check if convergence has been reached
		++iterations;

		// Visualize the results
		view(output, tgt, good_correspondences);

	} while (!converged);
	transform = final_transform;
	cout << "MSE: " << converged.getAbsoluteMSE() << endl;
}
```

Список используемых сокращений

SGM - Semi-Global Matching, алгоритм полуглобального сопоставления

FPFH - Fast Point Feature Histograms

PFH - Point Feature Histograms

ICP - Iterative Closest Point, итеративный алгоритм ближайших точек

PCL - Point Cloud Library

STL - Standard Template Library

DOF - degree of freedom

SOR - Statistical Outlier Removal filter, фильтр удаления выбросов на основе статистики

ROR - Radius Outlier Removal filter, фильтр удаления выбросов на основе радиуса

SVD - Singular Value Decomposition, сингулярное разложение матрицы

SDK - Software Development Kit